Screws.m
Robot kinematics package for Mathematica

Version 1.2
December 1992

Richard M. Murray
Division of Engineering and Applied Science
California Institute of Technology
Pasadena, CA 91125
murray@design.caltech.edu

Sudipto Sur
Division of Engineering and Applied Science
California Institute of Technology
Pasadena, CA 91125

1 Introduction

`Screws.m` is a Mathematica package for performing screw calculus. It follows the treatment described in *A Mathematical Introduction to Robotic Manipulation*, by R. M. Murray, Z. Li, and S. S. Sastry (CRC Press, 1994). This package implements screw theory in 3-dimensional Euclidean space (some functions work in n dimensions) and, when combined with the supplementary package `RobotLinks.m`, allows symbolic and numerical computation of the kinematics of open-chain robot manipulators as well as many other functions.

The `Screws` package is available via anonymous ftp from `avalon.caltech.edu` and may be used free of charge. Documentation and installation instructions are included with the source code for the package. The `Screws` package was written by R. Murray and S. Sur at the California Institute of Technology. All correspondence concerning the software should be sent to via e-mail to `murray@avalon.caltech.edu`. The authors assume no responsibility for the correctness or maintenance of the `Screws` package. The source code is currently available *only* via anonymous ftp.

The `Screws` package implements screw theory in 3-dimensional Euclidean space, R^3. It uses homogeneous coordinates to represent points, vectors, and rigid motions, making it easy to integrate into other Mathematica packages.

2 Rigid Body Motion

The Screws package consists of two groups of functions. The first group operates on rotation matrices and implements all of the mathematical operations described in Section 2 of Chapter 2 of MLS. The following functions are defined for computing in SO(3):

`AxisToSkew[w]`
: Generate a skew-symmetric matrix give a vector `w` in R^3

`RotationAxis[R]`
: Calculate the axis of rotation for a matrix `R` in SO(3).

`SkewExp[S, theta]`
: Calculate the exponential of a skew-symmetric matrix. If `theta` is not specified, it defaults to 1. If the first argument to `SkewExp` is a vector, `SkewExp` first converts it to a skew-symmetric matrix and then takes its exponential.

`SkewToAxis[S]`
: Generates a vector given a skew-symmetric matrix.

Limited error checking is used to insure that the arguments to the functions are in the proper form.

The second group of functions implements calculations on SE(3). Rigid body transformations are represented using 4x4 matrices. Functions are provided for transforming points and vectors to and from homogeneous coordinates, as well as converting a translation and rotation pair into a 4x4 matrix. The following functions are defined for use in SE(3):

`HomogeneousToTwist[xi]`
: Convert `xi` from a 4x4 matrix to a 6-vector.

`PointToHomogeneous[q]`
: Generate the homogeneous representation of a point `q` in R^3.

`RigidAdjoint[g]`
: Generate the adjoint matrix corresponding to `g`.

`RigidOrientation[g]`
: Extract the rotation matrix `R` from a homogeneous matrix `g`.

`RigidPosition[g]`
: Extract the position vector `p` from a homogeneous matrix `g`.

`RigidTwist[g]`
: Compute the twist `xi` in R^6 which generates the homogeneous matrix `g`.

`RPToHomogeneous[R,p]`
: Construct a 4x4 homogeneous matrix from a rotation matrix `R` and a translation `p`.

`ScrewToTwist[h, q, w]`
: Return the twist coordinates of a screw with pitch `h` through the point `q` and in the direction `w`. If `h == Infinity`, then a pure translational twist is generated. In this case, `q` is ignored and `w` gives the direction of translation.

`TwistAxis[xi]`
: Compute the axis of the screw corresponding to a twist. The axis is represented as a pair `{q, w}`, where `q` is a point on the axis and `w` is a unit vector describing the direction of the axis. The twist `xi` can be specified either as a 6-vector or a 4x4 matrix.

`TwistExp[xi, theta]`
: Compute the matrix exponential of a twist `xi`. The default value of `theta` is 1. If the first argument to `TwistExp` is a 6-vector, it is automatically converted to a 4x4 matrix.

`TwistPitch[xi]`
: Compute the pitch of a twist.

`TwistMagnitude[xi]`
: Compute the magnitude of a twist.

`TwistToHomogeneous[xi]`
: Convert `xi` from a 6-vector to a 4x4 matrix.

`VectorToHomogeneous[q]`
: Generate the homogeneous representation of a vector.

Limited error checking is used to insure that the arguments to the functions are in the proper form.

3 Robot Kinematics

The functions defined in the `Screws` package can be used to analyze the kinematics of a robot manipulator. This section describes this process and defines some new functions which streamline the analysis of manipulator kinematics. These functions are contained in the package '`RobotLinks.m`'.

The forward kinematics for a robot manipulator can be written as a product of exponentials (of twists). The following functions are defined for creating twists specifically for robot manipulators:

`RevoluteTwist[q, w]`
: Construct a twist corresponding to a revolute joint in the direction `w` going through the point `q`.

`PrismaticTwist[q, w]`
: Construct a twist corresponding to a prismatic joint in the direction `w` going through the point `q`.

Once the twists are defined, the forward kinematic map and the manipulator Jacobian can be calculated using matrix multiplication combined with the `TwistExp` and `RigidAdjoint` functions. These computations are automated by the following functions:

`ForwardKinematics[{xi1, th1}, {xi2, th2}, ..., gst0]`
: Compute the forward kinematics map using the product of exponentials formula. The pairs `{xi, th}` define the joint twist and joint angle (or displacement) for each joint of the manipulator.

`SpatialJacobian[{xi1, th1}, {xi2, th2}, ..., gst0]`
: Compute the spatial manipulator Jacobian for the manipulator. The pairs `{xi, th}` are given as in the `ForwardKinematics` function.

An example of the usage of `Screws` and `RobotLinks` packages is shown below for computing the kinematics of a SCARA manipulator.

```
<<Screws.m                          (* screws package      *)
<<RobotLinks.m                      (* additional functions *)

(* Twist axes for SCARA robot, starting from the base *)
xi1 = RevoluteTwist[{0,0,0},     {0,0,1}];      (* base  *)
xi2 = RevoluteTwist[{0,l1,0},    {0,0,1}];      (* elbow *)
xi3 = RevoluteTwist[{0,l1+l2,0}, {0,0,1}];      (* wrist *)
xi4 = PrismaticTwist[{0,0,0},    {0,0,1}];

(* Location of the tool frame at reference configuration *)
gst0 = RPToHomogeneous[IdentityMatrix[3], {0,l1+l2,0}];

(* Forward kinematics map *)
gst = Simplify[
  ForwardKinematics[
    {xi1,th1}, {xi2,th2}, {xi3,th3}, {xi4,th4}, gst0
```

```
  ]
];

(* Spatial manipulator Jacobian *)
Js  = Simplify[
  SpatialJacobian[{xi1,th1}, {xi2,th2}, {xi3,th3}, {xi4,th4}, gst0]
];
```

4 Reference

This section gives an alphabetical list of the commands and options defined in 'Simulate.m'.

- AxisSize

  AxisSize is an option to DrawFrame which sets the length of an axis.

- AxisToSkew

  AxisToSkew[w] generates a skew-symmetric matrix given a 3-vector w.

  AxisToSkew returns a 3x3 matrix which represents the cross product operator.

- BodyJacobian

  BodyJacobian[{xi1,th1},{xi2,th2},...,g0] computes the body manipulator Jacobian of a robot defined by the given twists.

  This function is part of the RobotLinks.m package.

- DrawFrame

  DrawScrew[q, w, h] generates a graphical description of a screw.

- DrawScrew

  DrawScrew[q, w, h] generates a graphical description of a screw.

- ForwardKinematics

  ForwardKinematics[{xi1,th1},...,{xiN,thN},g0] computes the forward kinematics via the product of exponentials formula.

- HomogeneousToTwist

  HomogeneousToTwist[xi] converts xi from a 4x4 matrix to a 6 vector.

- InertiaToCoriolis

  InertiaToCoriolis[M, theta] computes the Coriolis matrix given the inertia matrix, M, and a list of the joint variables, theta.

  This function is part of the RobotLinks.m package.

- PointToHomogeneous

  PointToHomogeneous[q] gives the homogeneous representation of a point.

  PointToHomogeneous converts a point in Euclidean space to its homogeneous representation by appending a '1' to the vector.

- `PrismaticTwist`

  `PrismaticTwist[q,w]` gives the 6-vector corresponding to point q on the axis and a screw with axis w for a prismatic joint.

  This function is part of the `RobotLinks.m` package.

- `RevoluteTwist`

  `RevoluteTwist[q,w]` gives the 6-vector corresponding to point q on the axis and a screw with axis w for a revolute joint.

  This function is part of the `RobotLinks.m` package.

- `RPToHomogeneous`

  `RPToHomogeneous[R,p]` forms homogeneous matrix from rotation matrix R and position vector p.

  `RPToHomogeneous` converts an element (R, p) in SE(3) into a 4x4 matrix.

- `RigidAdjoint`

  `RigidAdjoint[g]` gives the adjoint matrix corresponding to g.

  `RigidAdjoint` constructs a 6x6 matrix which represents the adjoint of the rigid transformation `g`. The rigid transformation `g` should be a 4x4 homogeneous matrix representing and element of SE(3). See ⟨undefined⟩ [RPToHomogeneous], page ⟨undefined⟩

- `RigidInverse`

  `RigidInverse[g]` gives the inverse transformation of g.

- `RigidOrientation`

  `RigidOrientation[g]` extracts the rotation matrix `R` from `g`.

  `RigidOrientation` extracts the rotation component of a rigid motion from the 4x4 homogeneous matrix `g`. See ⟨undefined⟩ [RigidPosition], page ⟨undefined⟩.

- `RigidPosition`

  `RigidPosition[g]` extracts the position vector `p` from `g`.

  `RigidPosition` extracts the translational component of a rigid motion from the 4x4 homogeneous matrix `g`. See ⟨undefined⟩ [RigidOrientation], page ⟨undefined⟩.

- `RigidTwist`

  `RigidTwist[g]` returns an equivalent twist given a rigid motion `g`.

RigidTwist calculates a twist which generates the rigid motion g. This twist is not unique.

- RotationAxis

  RotationAxis[R] finds the rotation axis for a rotation matrix.

  RotationAxis finds an equivalent axis for a given rotation matrix.

- RotationQ

  RotationQ[M] tests wether a matrix M is a rotation matrix.

  RotationQ checks to see if M is a 3x3 matrix which satisfies Transpose[M] . M == IdentityMatrix[3] and Det[M] == 1. RotationQ may return False for non-numeric matrices.

- ScrewSize

  ScrewSize is an option for DrawScrew which sets the length of a screw.

- ScrewToTwist

  ScrewToTwist[h, q, w] returns the twist coordinates of a screw.

- SkewExp

  SkewExp[w, theta] gives the matrix exponential of an axis w. Default value of theta is 1.

  SkewExp uses Rodriguez's formula to calculate the matrix exponential of a skew-symmetric matrix. w can either be a 3-vector or a skew-symmetrix matrix.

- SkewToAxis

  SkewToAxis[S] extracts a vector from a skew-symmetric matrix S.

  SkewToAxis extracts a 3-vector from a 3x3 skew-symmetric matrix.

- SpatialJacobian

  SpatialJacobian[{xi1,th1},{xi2,th2},...,g0] computes the spatial manipulator Jacobian of a robot defined by the given twists.

  This function is part of the RobotLinks.m package.

- StackCols

  StackCols[mat1, mat2, ...] stacks matrix columns together

- StackRows

  StackRows[mat1, mat2, ...] stacks matrix rows together

- `TwistAxis`

    `TwistAxis[xi]` gives axis of a screw corresponding to a twist.

- `TwistExp`

    `TwistExp[xi, theta]` gives the matrix exponential of a twist `xi`. Default value of `theta` is 1.

    `TwistExp` computes the matrix exponential of a twist. The twist may be specified as either a 6-vector (which will be converted to a 4x4 matrix with `TwistToHomogeneous`) or a 4x4 twist matrix.

- `TwistMagnitude`

    `TwistMagnitude[xi]` returns the magnitude of a twist.

- `TwistPitch`

    `TwistPitch[xi]` returns the pitch of a twist.

    `TwistPitch` returns the pitch of a twist vector or matrix. An inifinite pitch twist returns `Inifinity`.

- `TwistToHomogeneous`

    `TwistToHomogeneous[xi]` converts `xi` from a 6 vector to a 4X4 matrix.

    `TwistToHomogeneous` converts a twist to its 4x4 homogeneous representation.

- `VectorToHomogeneous`

    `VectorToHomogeneous[q]` gives the homogeneous representation of a point.

    `VectorToHomogeneous` converts a point in Euclidean space to its homogeneous representation by appending a ‘1’ to the vector.

Index