
kinematics Documentation

Release 0.1.7

Shankar Kulumani

Dec 08, 2018

Contents:

1	Kinematics Package	1
2	attitude module	3
2.1	Requirements	3
3	sphere module	9
4	Indices and tables	11
	Python Module Index	13

CHAPTER 1

Kinematics Package

Attitude Kinematics - transformations between attitude representations

This module provides a variety of functions to perform attitude kinematics. It transforms between a variety of attitude representations, provides functions for working with attitude,

Example

Provide some examples of how to use this module

References

[1] M. D. Shuster, A survey of attitude representations, Journal of the Astronautical Sciences, vol. 41, no. 8, pp. 439-517, 1993. [2] P. C. Hughes, Spacecraft Attitude Dynamics. Dover Publications, 2004. [3] J. R. Wertz, Spacecraft Attitude Determination and Control, vol. 73. Springer, 1978.

2.1 Requirements

List all the required components for this library to work properly.

`kinematics.attitude.ang_veltoaxisangledot` (*angle, axis, Omega*)

Compute kinematics for axis angle representation

`kinematics.attitude.ang_veltodcmdot` (*R, Omega*)

Convert angular velocity to DCM dot Omega - angular velocity defined in body frame

`kinematics.attitude.axisangledottoang_vel` (*angle, axis, angle_dot, axis_dot*)

Convert axis angle representation to angular velocity in body frame

`kinematics.attitude.axisangletodcm` (*angle, axis*)

`kinematics.attitude.body313dot` (*theta, Omega*)

`kinematics.attitude.body313dot_to_ang_vel(theta, theta_dot)`

`kinematics.attitude.body313todcm(theta)`

`kinematics.attitude.dcm2body313(dcm)`

Convert DCM to body Euler 3-1-3 angles

`kinematics.attitude.dcmdottoang_vel(R, Rdot)`

Convert a rotation matrix to angular velocity

w - angular velocity in inertial frame Omega - angular velocity in body frame

`kinematics.attitude.dcmtoaxisangle(R)`

`kinematics.attitude.dcmtoquat(dcm)`

Convert DCM to quaternion

This function will convert a rotation matrix, also called a direction cosine matrix into the equivalent quaternion.

dcm - (3,3) numpy array Numpy rotation matrix which defines a rotation from the b to a frame

quat - (4,) numpy array Array defining a quaterion where the quaternion is defined in terms of a vector and a scalar part. The vector is related to the eigen axis and equivalent in both reference frames [x y z w]

`kinematics.attitude.hat_map(vec)`

Return that hat map of a vector

Inputs: vec - 3 element vector

Outputs: skew - 3,3 skew symmetric matrix

`kinematics.attitude.normalize(num_in, lower=0, upper=360, b=False)`

Normalize number to range [lower, upper) or [lower, upper].

Parameters

- **num** (*float*) – The number to be normalized.
- **lower** (*int*) – Lower limit of range. Default is 0.
- **upper** (*int*) – Upper limit of range. Default is 360.
- **b** (*bool*) – Type of normalization. Default is False. See notes.

When b=True, the range must be symmetric about 0. When b=False, the range must be symmetric about 0 or lower must be equal to 0.

Returns n – A number in the range [lower, upper) or [lower, upper].

Return type float

Raises ValueError – If lower >= upper.

Notes

If the keyword *b* == *False*, then the normalization is done in the following way. Consider the numbers to be arranged in a circle, with the lower and upper ends sitting on top of each other. Moving past one limit, takes the number into the beginning of the other end. For example, if range is [0 - 360), then 361 becomes 1 and 360 becomes 0. Negative numbers move from higher to lower numbers. So, -1 normalized to [0 - 360) becomes 359.

When b=False range must be symmetric about 0 or lower=0.

If the keyword `b == True`, then the given number is considered to “bounce” between the two limits. So, -91 normalized to [-90, 90], becomes -89, instead of 89. In this case the range is [lower, upper]. This code is based on the function `fnt_delta` of *TPM*.

When `b=True` range must be symmetric about 0.

Examples

```
>>> normalize(-270,-180,180)
90.0
>>> import math
>>> math.degrees(normalize(-2*math.pi,-math.pi,math.pi))
0.0
>>> normalize(-180, -180, 180)
-180.0
>>> normalize(180, -180, 180)
-180.0
>>> normalize(180, -180, 180, b=True)
180.0
>>> normalize(181,-180,180)
-179.0
>>> normalize(181, -180, 180, b=True)
179.0
>>> normalize(-180,0,360)
180.0
>>> normalize(36,0,24)
12.0
>>> normalize(368.5,-180,180)
8.5
>>> normalize(-100, -90, 90)
80.0
>>> normalize(-100, -90, 90, b=True)
-80.0
>>> normalize(100, -90, 90, b=True)
80.0
>>> normalize(181, -90, 90, b=True)
-1.0
>>> normalize(270, -90, 90, b=True)
-90.0
>>> normalize(271, -90, 90, b=True)
-89.0
```

`kinematics.attitude.quatdot` (*quat*, *Omega*)

`kinematics.attitude.quatdot_ang_vel` (*quat*, *quat_dot*)

`kinematics.attitude.quattodcm` (*quat*)

Convert quaternion to DCM

This function will convert a quaternion to the equivalent rotation matrix, or direction cosine matrix.

quat - (4,) numpy array Array defining a quaterion where the quaternion is defined in terms of a vector and a scalar part. The vector is related to the eigen axis and equivalent in both reference frames [x y z w]

dcm - (3,3) numpy array Numpy rotation matrix which defines a rotation from the b to a frame

`kinematics.attitude.rot1` (*angle*, *form=u'c'*)

Euler rotation about first axis

This computes the rotation matrix associated with a rotation about the first axis. It will output matrices assuming column or row format vectors.

For example, to transform a vector from reference frame b to reference frame a:

Column Vectors : $a = \text{rot1}(\text{angle}, 'c').\text{dot}(b)$ Row Vectors : $a = b.\text{dot}(\text{rot1}(\text{angle}, 'r'))$

It should be clear that $\text{rot1}(\text{angle}, 'c') = \text{rot1}(\text{angle}, 'r').T$

Parameters

- **angle** (*float*) – Angle of rotation about first axis. In radians
- **form** (*str*) – Flag to choose between row or column vector convention.

Returns **mat** – Rotation matrix

Return type `numpy.ndarray` of shape (3,3)

`kinematics.attitude.rot2 (angle, form='c')`

Euler rotation about second axis

This computes the rotation matrix associated with a rotation about the first axis. It will output matrices assuming column or row format vectors.

For example, to transform a vector from reference frame b to reference frame a:

Column Vectors : $a = \text{rot2}(\text{angle}, 'c').\text{dot}(b)$ Row Vectors : $a = b.\text{dot}(\text{rot1}(\text{angle}, 'r'))$

It should be clear that $\text{rot2}(\text{angle}, 'c') = \text{rot2}(\text{angle}, 'r').T$

Parameters

- **angle** (*float*) – Angle of rotation about second axis. In radians
- **form** (*str*) – Flag to choose between row or column vector convention.

Returns **mat** – Rotation matrix

Return type `numpy.ndarray` of shape (3,3)

`kinematics.attitude.rot3 (angle, form='c')`

Euler rotation about thrid axis

This computes the rotation matrix associated with a rotation about the third axis. It will output matrices assuming column or row format vectors.

For example, to transform a vector from reference frame b to reference frame a:

Column Vectors : $a = \text{rot3}(\text{angle}, 'c').\text{dot}(b)$ Row Vectors : $a = b.\text{dot}(\text{rot3}(\text{angle}, 'r'))$

It should be clear that $\text{rot3}(\text{angle}, 'c') = \text{rot3}(\text{angle}, 'r').T$

Parameters

- **angle** (*float*) – Angle of rotation about first axis. In radians
- **form** (*str*) – Flag to choose between row or column vector convention.

Returns **mat** – Rotation matrix

Return type `numpy.ndarray` of shape (3,3)

`kinematics.attitude.test_rot_mat_in_special_orthogonal_group (R)`

`kinematics.attitude.unit_vector (q)`

Unit vector in direction of q

`qhat = unit_vector(q)`

Parameters \mathbf{q} ((n,) *numpy array*) – Vector in \mathbb{R}^n

Returns $\mathbf{\hat{q}}$ – The unit vector in the direction of \mathbf{q}

Return type (n,) *numpy array*

Notes

You may include some math:

$$\hat{q} = \frac{q}{|q|}$$

Shankar Kulumani GWU skulumani@gwu.edu

`kinematics.attitude.vee_map(skew)`

Return the vee map of a vector

CHAPTER 3

sphere module

n-Sphere operations

This module holds some useful functions for dealing with elements of n-spheres. Most usually we tend to deal with the 1-sphere and the 2-sphere.

`kinematics.sphere.perturb_vec(q, cone_half_angle=2)`

Perturb a vector randomly

`qp = perturb_vec(q, cone_half_angle=2)`

Parameters

- **q** ((n,) numpy array) – Vector to perturb
- **cone_half_angle** (float) – Maximum angle to perturb the vector in degrees

Returns

- **perturbed** ((n,) numpy array) – Perturbed numpy array
- *Author*
- *—*
- *Shankar Kulumani GWU skulumani@gwu.edu*

References

`kinematics.sphere.rand(n, **kwargs)`

Random vector from the n-Sphere

This function will return a random vector which is an element of the n-Sphere. The n-Sphere is defined as a vector in R^{n+1} with a norm of one.

Basically, we'll find a random vector in R^{n+1} and normalize it. This uses the method of Marsaglia 1972.

Parameters None –

Returns Random (n+1,) numpy vector with a norm of 1

Return type rvec

`kinematics.sphere.tan_rand(q, seed=9)`

Find a random vector in the tangent space of the n sphere

This function will find a random orthogonal vector to q.

Parameters \mathbf{q} – (n+1,) array which is in the n-sphere

Returns (n+1,) array which is orthogonal to n-sphere and also random

Return type qd

CHAPTER 4

Indices and tables

- `genindex`
- `modindex`
- `search`

k

`kinematics.attitude`, 3

`kinematics.sphere`, 9

A

ang_veltoaxisangledot() (in module kinematics.attitude), 3
ang_veltodcmdot() (in module kinematics.attitude), 3
axisangledottoang_vel() (in module kinematics.attitude), 3
axisangletodcm() (in module kinematics.attitude), 3

B

body313dot() (in module kinematics.attitude), 3
body313dot_to_ang_vel() (in module kinematics.attitude), 3
body313todcm() (in module kinematics.attitude), 4

D

dcm2body313() (in module kinematics.attitude), 4
dcmdottoang_vel() (in module kinematics.attitude), 4
dcmttoaxisangle() (in module kinematics.attitude), 4
dcmttoquat() (in module kinematics.attitude), 4

H

hat_map() (in module kinematics.attitude), 4

K

kinematics.attitude (module), 3
kinematics.sphere (module), 9

N

normalize() (in module kinematics.attitude), 4

P

perturb_vec() (in module kinematics.sphere), 9

Q

quatdot() (in module kinematics.attitude), 5
quatdot_ang_vel() (in module kinematics.attitude), 5
quattodcm() (in module kinematics.attitude), 5

R

rand() (in module kinematics.sphere), 9
rot1() (in module kinematics.attitude), 5
rot2() (in module kinematics.attitude), 6
rot3() (in module kinematics.attitude), 6

T

tan_rand() (in module kinematics.sphere), 10
test_rot_mat_in_special_orthogonal_group() (in module kinematics.attitude), 6

U

unit_vector() (in module kinematics.attitude), 6

V

vee_map() (in module kinematics.attitude), 7