skrobot Documentation

Iori Yanokura, Kentaro Wada

CONTENTS

1	Insta	Allation and Setup Guide	3
	1.1	Python Installation	
2	Usag	ge Examples	5
	2.1	Loading from a File	5
	2.2	Visualization	5
	2.3	Accessing Links and Joints	5
	2.4	Inverse Kinematics	6
3	API	Reference	7
	3.1	Coordinates	7
	3.2	Models	39
	3.3	Functions	93
	3.4	Interfaces	119
	3.5	Signed distance function (SDF)	128
	3.6	Planning	137
4	Deve	elopment Guide	143
	4.1	Setting Up	143
	4.2	Running Code Style Checks	
	4.3	Running Tests	144
	4.4	Building Documentation	144
5	Indic	ces and tables	145
Ру	thon l	Module Index	147
In	dex		149

Scikit-Robot is a simple pure-Python library for loading, manipulating, and visualizing URDF files and robot specifications. For example, here's a rendering of a PR2 robot moving after being loaded by this library.

CONTENTS 1

2 CONTENTS

CHAPTER

ONE

INSTALLATION AND SETUP GUIDE

1.1 Python Installation

This package is pip-installable for any Python version. Simply run the following command:

pip install scikit-robot

1.2 Installing in Development Mode

If you're planning on contributing to this repository, please see the *Development Guide*.

CHAPTER

TWO

USAGE EXAMPLES

This page documents several simple use cases for you to try out. For full details, see the *API Reference*, and check out the full class reference for *RobotModel*.

2.1 Loading from a File

You can load a URDF from any .urdf file, as long as you fix the links to be relative or absolute links rather than ROS resource URLs.

2.2 Visualization

```
>>> viewer = skrobot.viewers.TrimeshSceneViewer(resolution=(640, 480))
>>> viewer.add(robot_model)
>>> viewer.show()
```

If you would like to update renderer:

```
>>> viewer.redraw()
```

2.3 Accessing Links and Joints

You have direct access to link and joint information.

```
>>> for link in robot_model.link_list:
... print(link.name)
```

```
>>> for joint in robot_model.joint_list:
... print(joint.name)
```

```
>>> robot_model.l_elbow_flex_joint.joint_angle()
0.0
```

```
>>> robot_model.l_elbow_flex_joint.joint_angle(-0.5)
-0.5
```

```
>>> robot_model.l_elbow_flex_joint.joint_angle()
-0.5
```

2.4 Inverse Kinematics

First, set the initial pose. Note that the position of the prismatic joint is in [m] and angles of rotational joints are in [rad].

```
>>> robot_model.torso_lift_joint.joint_angle(0.05)
>>> robot_model.l_shoulder_pan_joint.joint_angle(60 * 3.14/180.0)
>>> robot_model.l_shoulder_lift_joint.joint_angle(74 * 3.14/180.0)
>>> robot_model.l_upper_arm_roll_joint.joint_angle(70* 3.14/180.0)
>>> robot_model.l_elbow_flex_joint.joint_angle(-120 * 3.14/180.0)
>>> robot_model.l_forearm_roll_joint.joint_angle(20 * 3.14/180.0)
>>> robot_model.l_wrist_flex_joint.joint_angle(-30 * 3.14/180.0)
>>> robot_model.l_wrist_roll_joint.joint_angle(180 * 3.14/180.0)
>>> robot_model.r_shoulder_pan_joint.joint_angle(-60 * 3.14/180.0)
>>> robot_model.r_shoulder_lift_joint.joint_angle(74 * 3.14/180.0)
>>> robot_model.r_upper_arm_roll_joint.joint_angle(-70 * 3.14/180.0)
>>> robot_model.r_elbow_flex_joint.joint_angle(-120 * 3.14/180.0)
>>> robot_model.r_forearm_roll_joint.joint_angle(-20 * 3.14/180.0)
>>> robot_model.r_wrist_flex_joint.joint_angle(-30 * 3.14/180.0)
>>> robot_model.r_wrist_roll_joint.joint_angle(180 * 3.14/180.0)
>>> robot_model.head_pan_joint.joint_angle(0)
>>> robot_model.head_tilt_joint.joint_angle(0)
```

Next, set move_target and link_list

Set target_coords.

```
>>> target_coords = skrobot.coordinates.Coordinates([0.5, -0.3, 0.7], [0, 0, 0])
>>> robot_model.inverse_kinematics(
... target_coords,
... link_list=link_list,
... move_target=move_target)
```

CHAPTER

THREE

API REFERENCE

3.1 Coordinates

3.1.1 Coordinates classes

skrobot.coordinates.Coordinates

Coordinates class to manipulate rotation and translation.

skrobot.coordinates.CascadedCoords

skrobot.coordinates.Coordinates

Coordinates class to manipulate rotation and translation.

Parameters

- **pos** (*list or numpy.ndarray or None*) shape of (3,) translation vector. or 4x4 homogeneous transformation matrix. If the homogeneous transformation matrix is given, *rot* will be overwritten. If this value is *None*, set [0, 0, 0] vector as default.
- **rot** (*list or numpy.ndarray or None*) we can take 3x3 rotation matrix or [yaw, pitch, roll] or quaternion [w, x, y, z] order If this value is *None*, set the identity matrix as default.
- name (str or None) name of this coordinates
- **check_validity** (*bool* (*optional*)) Default *True*. If this value is *True*, check whether an input rotation and an input translation are valid.

Methods

T()

Return 4x4 homogeneous transformation matrix.

Returns matrix – homogeneous transformation matrix shape of (4, 4)

Return type numpy.ndarray

```
>>> from numpy import pi
>>> from skrobot.coordinates import make_coords
>>> c = make_coords()
>>> c.T()
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
>>> c.translate([0.1, 0.2, 0.3])
>>> c.rotate(pi / 2.0, 'y')
array([[ 2.22044605e-16, 0.00000000e+00,
                                            1.00000000e+00,
         1.00000000e-01],
       [0.000000000e+00, 1.00000000e+00,
                                            0.00000000e+00,
         2.00000000e-01],
       [-1.000000000e+00, 0.00000000e+00,
                                            2.22044605e-16,
         3.00000000e-01],
       [ 0.00000000e+00, 0.00000000e+00,
                                            0.00000000e+00,
         1.00000000e+00]])
```

axis(ax)

changed()

Return False

This is used for CascadedCoords compatibility

Returns False – always return False

Return type bool

coords()

Return a deep copy of the Coordinates.

copy()

Return a deep copy of the Coordinates.

copy_coords()

Return a deep copy of the Coordinates.

copy_worldcoords()

Return a deep copy of the Coordinates.

difference_position(coords, translation_axis=True)

Return differences in position of given coords.

Parameters

- coords (skrobot.coordinates.Coordinates) given coordinates
- translation_axis (str or bool or None (optional)) we can take 'x', 'y', 'z', 'xy', 'yz', 'zx', 'xx', 'yy', 'zz', True or False(None).

Returns dif_pos – difference position of self coordinates and coords considering translation axis.

Return type numpy.ndarray

```
>>> from skrobot.coordinates import Coordinates
>>> from skrobot.coordinates import transform_coords
>>> from numpy import pi
>>> c1 = Coordinates().translate([0.1, 0.2, 0.3]).rotate(
             pi / 3.0, 'x')
>>> c2 = Coordinates().translate([0.3, -0.3, 0.1]).rotate(
            pi / 2.0, 'y')
>>> c1.difference_position(c2)
array([ 0.2
            , -0.42320508, 0.3330127 ])
>>> c1 = Coordinates().translate([0.1, 0.2, 0.3]).rotate(0, 'x')
>>> c2 = Coordinates().translate([0.3, -0.3, 0.1]).rotate(
             pi / 3.0, 'x')
. . .
>>> c1.difference_position(c2)
array([ 0.2, -0.5, -0.2])
```

difference_rotation(coords, rotation_axis=True)

Return differences in rotation of given coords.

Parameters

- coords (skrobot.coordinates.Coordinates) given coordinates
- rotation_axis(str or bool or None (optional)) we can take 'x', 'y', 'z', 'xx', 'yy', 'zz', 'xm', 'ym', 'zm', 'xy', 'yx', 'yz', 'zy', 'zx', True or False(None).

Returns dif_rot – difference rotation of self coordinates and coords considering rotation_axis.

Return type numpy.ndarray

Examples

```
>>> from numpy import pi
>>> from skrobot.coordinates import Coordinates
>>> from skrobot.coordinates.math import rpy_matrix
>>> coord1 = Coordinates()
>>> coord2 = Coordinates(rot=rpy_matrix(pi / 2.0, pi / 3.0, pi / 5.0))
>>> coord1.difference_rotation(coord2)
array([-0.32855112, 1.17434985, 1.05738936])
>>> coord1.difference_rotation(coord2, rotation_axis=False)
array([0, 0, 0])
>>> coord1.difference_rotation(coord2, rotation_axis='x')
                , 1.36034952, 0.78539816])
>>> coord1.difference_rotation(coord2, rotation_axis='y')
                           , 0.97442695])
array([0.35398131, 0.
>>> coord1.difference_rotation(coord2, rotation_axis='z')
array([-0.88435715, 0.74192175, 0.
```

Using mirror option ['xm', 'ym', 'zm'], you can allow differences of mirror direction.

```
>>> coord1 = Coordinates()
>>> coord2 = Coordinates().rotate(pi, 'x')
>>> coord1.difference_rotation(coord2, 'xm')
```

(continues on next page)

(continued from previous page)

```
array([-2.99951957e-32, 0.00000000e+00, 0.00000000e+00])
>>> coord1 = Coordinates()
>>> coord2 = Coordinates().rotate(pi / 2.0, 'x')
>>> coord1.difference_rotation(coord2, 'xm')
array([-1.57079633, 0. , 0. ])
```

disable_hook()

get_transform()

Return Transform object

Returns transform – corrensponding Transform to this coordinates

Return type skrobot.coordinates.base.Transform

inverse_rotate_vector(v)

inverse_transform_vector(vec)

Transform vector in world coordinates to local coordinates

Parameters vec (numpy.ndarray) – 3d vector. We can take batch of vector like (batch_size, 3)

Returns transformed point – transformed point

Return type numpy.ndarray

inverse_transformation(dest=None)

Return a invese transformation of this coordinate system.

Create a new coordinate with inverse transformation of this coordinate system.

$$\left(\begin{array}{cc} R^{-1} & -R^{-1}p \\ 0 & 1 \end{array}\right)$$

Parameters dest (*None or* skrobot.coordinates.Coordinates) – If dest is given, the result of transformation is in-placed to dest.

Returns dest – result of inverse transformation.

Return type skrobot.coordinates.Coordinates

move_coords(target_coords, local_coords)

Transform this coordinate so that local_coords to target_coords.

Parameters

- target_coords (skrobot.coordinates.Coordinates) target coords.
- local_coords (skrobot.coordinates.Coordinates) local coords to be aligned.

Returns self.worldcoords() – world coordinates.

Return type skrobot.coordinates.Coordinates

newcoords(c, pos=None, check_validity=True)

Update of coords is always done through newcoords.

Parameters

- **c** (skrobot.coordinates.Coordinates *or numpy.ndarray*) If pos is *None*, *c* means new Coordinates. If pos is given, *c* means rotation matrix.
- **pos** (numpy.ndarray or None) new translation.

• **check_validity** (*boo1*) – If this value is *True*, check whether an input rotation and an input translation are valid.

```
orient_with_matrix(rotation_matrix, wrt='world')
```

Force update this coordinate system's rotation.

Parameters

- **rotation_matrix** (*numpy.ndarray*) 3x3 rotation matrix.
- wrt (str or skrobot.coordinates.Coordinates) reference coordinates.

```
parent_orientation(v, wrt)
```

```
rotate(theta, axis=None, wrt='local')
```

Rotate this coordinate by given theta and axis.

This coordinate system is rotated relative to theta radians around the *axis* axis. Note that this function does not change a position of this coordinate. If you want to rotate this coordinates around with world frame, you can use *transform* function. Please see examples.

Parameters

- **theta** (*float*) relartive rotation angle in radian.
- axis (str or None or numpy.ndarray) axis of rotation. The value of axis is represented as wrt frame.
- wrt (str or skrobot.coordinates.Coordinates) -

Returns self

Return type skrobot.coordinates.Coordinates

Examples

```
>>> from skrobot.coordinates import Coordinates
>>> from numpy import pi
>>> c = Coordinates()
>>> c.translate((1.0, 0, 0))
>>> c.rotate(pi / 2.0, 'z', wrt='local')
>>> c.translation
array([1., 0., 0.])
```

```
>>> c.transform(Coordinates().rotate(np.pi / 2.0, 'z'), wrt='world')
>>> c.translation
array([0., 1., 0.])
```

rotate_vector(v)

Rotate 3-dimensional vector using rotation of this coordinate

Parameters v (numpy.ndarray) – vector shape of (3,)

Returns np.matmul(self.rotation, v) – rotated vector

Return type numpy.ndarray

```
>>> from skrobot.coordinates import Coordinates
>>> from numpy import pi
>>> c = Coordinates().rotate(pi, 'z')
>>> c.rotate_vector([1, 2, 3])
array([-1., -2., 3.])
```

rotate_with_matrix(mat, wrt='local')

Rotate this coordinate by given rotation matrix.

This is a subroutine of self.rotate function.

Parameters

- mat (numpy.ndarray) rotation matrix shape of (3, 3)
- wrt(str or skrobot.coordinates.Coordinates) with respect to.

Returns self

Return type skrobot.coordinates.Coordinates

rpy_angle()

Return a pair of rpy angles of this coordinates.

```
Returns rpy_angle(self.rotation) – a pair of rpy angles. See also skrobot.coordinates.math.rpy_angle
```

Return type tuple(numpy.ndarray, numpy.ndarray)

Examples

```
>>> import numpy as np
>>> from skrobot.coordinates import Coordinates
>>> c = Coordinates().rotate(np.pi / 2.0, 'x').rotate(np.pi / 3.0, 'z')
>>> r.rpy_angle()
(array([ 3.84592537e-16, -1.04719755e+00, 1.57079633e+00]),
array([ 3.14159265, -2.0943951 , -1.57079633]))
```

transform(c, wrt='local', out=None)

Transform this coordinates by coords based on wrt

Note that this function changes this coordinates translation and rotation. If you would like not to change this coordinates, Please use copy_worldcoords() or give *out*.

Parameters

- c (skrobot.coordinates.Coordinates) coordinate
- wrt(str or skrobot.coordinates.Coordinates) If wrt is 'local' or self, multiply c from the right. If wrt is 'world' or 'parent' or self.parent, transform c with respect to worldcoord. If wrt is Coordinates, transform c with respect to c.
- **out** (*None or* skrobot.coordinates.Coordinates) If the *out* is specified, set new coordinates to *out*. Note that if the *out* is given, these coordinates don't change.

Returns self – return this coordinate

Return type skrobot.coordinates.Coordinates

transform_vector(v)

"Return vector represented at world frame.

Vector v given in the local coords is converted to world representation.

Parameters v (numpy.ndarray) – 3d vector. We can take batch of vector like (batch_size, 3)

Returns transformed_point - transformed point

Return type numpy.ndarray

transformation(c2, wrt='local')

```
translate(vec, wrt='local')
```

Translate this coordinates.

Note that this function changes this coordinates self. So if you don't want to change this class, use copy_worldcoords()

Parameters

- **vec** (list or numpy.ndarray) shape of (3,) translation vector. unit is [m] order.
- wrt (str or Coordinates (optional)) translate with respect to wrt.

Examples

```
>>> import numpy as np
>>> from skrobot.coordinates import Coordinates
>>> c = Coordinates()
>>> c.translation
array([0., 0., 0.], dtype=float32)
>>> c.translate([0.1, 0.2, 0.3])
>>> c.translation
array([0.1, 0.2, 0.3], dtype=float32)
```

```
>>> c = Coordinates()
>>> c.copy_worldcoords().translate([0.1, 0.2, 0.3])
>>> c.translation
array([0., 0., 0.], dtype=float32)
```

```
>>> c = Coordinates().rotate(np.pi / 2.0, 'y')
>>> c.translate([0.1, 0.2, 0.3])
>>> c.translation
array([ 0.3,  0.2, -0.1])
>>> c = Coordinates().rotate(np.pi / 2.0, 'y')
>>> c.translate([0.1, 0.2, 0.3], 'world')
>>> c.translation
array([0.1, 0.2, 0.3])
```

worldcoords()

Return thisself

worldpos()

Return translation of this coordinate

```
See also skrobot.coordinates.Coordinates.translation
```

```
Returns self.translation – translation of this coordinate
```

Return type numpy.ndarray

```
worldrot()
```

Return rotation of this coordinate

See also skrobot.coordinates.Coordinates.rotation

Returns self.rotation – rotation matrix of this coordinate

Return type numpy.ndarray

```
__eq__(value,/)
Return self==value.
__ne__(value,/)
Return self!=value.
__lt__(value,/)
Return self<value.
__le__(value,/)
Return self<=value.
__gt__(value,/)
Return self>value.
__ge__(value,/)
Return self>=value.
```

Return Transformed Coordinates.

Note that this function creates new Coordinates and does not change translation and rotation, unlike transform function.

```
\textbf{Parameters other\_c} \ (\texttt{skrobot.coordinates}. \texttt{Coordinates}) - \texttt{input coordinates}.
```

Returns out – transformed coordinates multiplied other_c from the right. $T = T_{self}$ T_{other_c} .

Return type *skrobot.coordinates.Coordinates*

```
__pow__(exponent)
```

__mul__(other_c)

Return exponential homogeneous matrix.

If exponent equals -1, return inverse transformation of this coords.

Parameters exponent (*numbers*. *Number*) – exponent value. If exponent equals -1, return inverse transformation of this coords. In current, support only -1 case.

```
Returns out – output.
```

Return type skrobot.coordinates.Coordinates

Attributes

dimension

Return dimension of this coordinate

Returns len(self.translation) – dimension of this coordinate

Return type int

dual_quaternion

Property of DualQuaternion

Return DualQuaternion representation of this coordinate.

Returns DualQuaternion - DualQuaternion representation of this coordinate

 $\textbf{Return type} \ \textit{skrobot.coordinates.dual_quaternion.} Dual \textit{Quaternion}$

name

Return this coordinate's name

Returns self. name – name of this coordinate

Return type str

quaternion

Property of quaternion

```
Returns \mathbf{q} - [\mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z}] quaternion
```

Return type numpy.ndarray

Examples

```
>>> from numpy import pi
>>> from skrobot.coordinates import make_coords
>>> c = make_coords()
>>> c.quaternion
array([1., 0., 0., 0.])
>>> c.rotate(pi / 3, 'y').rotate(pi / 5, 'z')
>>> c.quaternion
array([0.8236391 , 0.1545085 , 0.47552826, 0.26761657])
```

rotation

Return rotation matrix of this coordinates.

Returns self._rotation – 3x3 rotation matrix

Return type numpy.ndarray

translation

Return translation of this coordinates.

```
Returns self._translation – vector shape of (3, ). unit is [m] Return type numpy.ndarray
```

Examples

```
>>> from skrobot.coordinates import Coordinates
>>> c = Coordinates()
>>> c.translation
array([0., 0., 0.])
>>> c.translate([0.1, 0.2, 0.3])
>>> c.translation
array([0.1, 0.2, 0.3])
```

x_axis

Return x axis vector of this coordinates.

```
Returns axis – x axis.
```

Return type numpy.ndarray

y_axis

Return y axis vector of this coordinates.

```
Returns axis – y axis.
```

Return type numpy.ndarray

z_axis

Return z axis vector of this coordinates.

```
Returns axis – z axis.
```

Return type numpy.ndarray

skrobot.coordinates.CascadedCoords

class skrobot.coordinates.CascadedCoords(parent=None, *args, **kwargs)

Methods

T()

Return 4x4 homogeneous transformation matrix.

Returns matrix – homogeneous transformation matrix shape of (4, 4)

Return type numpy.ndarray

Examples

```
>>> from numpy import pi
>>> from skrobot.coordinates import make_coords
>>> c = make_coords()
>>> c.T()
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]
>>> c.translate([0.1, 0.2, 0.3])
>>> c.rotate(pi / 2.0, 'y')
array([[ 2.22044605e-16, 0.00000000e+00,
                                            1.00000000e+00,
         1.00000000e-01],
       [ 0.00000000e+00,
                          1.00000000e+00.
                                            0.00000000e+00.
         2.00000000e-01],
       [-1.000000000e+00, 0.00000000e+00,
                                            2.22044605e-16,
         3.00000000e-01],
       [ 0.00000000e+00, 0.00000000e+00,
                                           0.00000000e+00,
         1.00000000e+00]])
```

assoc(child, relative_coords='world', force=False, **kwargs)

Associate child coords to this coordinate system.

If *relative_coords* is *None* or 'world', the translation and rotation of childcoords in the world coordinate system do not change. If *relative_coords* is specified, childcoords is assoced at translation and rotation of *relative_coords*. By default, if child is already assoced to some other coords, raise an exception. But if *force* is *True*, you can overwrite the existing assoc relation.

Parameters

- child (CascadedCoords) child coordinates.
- relative_coords (None or Coordinates or str) child coordinates' relative coordinates.
- **force** (*bool*) predicate for overwriting the existing assoc-relation

Returns child – assoced child.

Return type CascadedCoords

```
>>> from skrobot.coordinates import CascadedCoords
>>> parent_coords = CascadedCoords(pos=[1, 0, 0])
>>> child_coords = CascadedCoords(pos=[1, 1, 0])
>>> parent_coords.assoc(child_coords)
#<CascadedCoords 0x7f1d30e29510 1.000 1.000 0.000 / 0.0 -0.0 0.0>
>>> child_coords.worldpos()
array([1., 1., 0.])
>>> child_coords.translation
array([0., 1., 0.])
```

None and 'world' have the same meaning.

```
>>> parent_coords = CascadedCoords(pos=[1, 0, 0])
>>> child_coords = CascadedCoords(pos=[1, 1, 0])
>>> parent_coords.assoc(child_coords, relative_coords='world')
#<CascadedCoords 0x7f1d30e29510 1.000 1.000 0.000 / 0.0 -0.0 0.0>
>>> child_coords.worldpos()
array([1., 1., 0.])
```

If *relative_coords* is 'local', *child* is associated at world translation and world rotation of *child* from this coordinate system.

```
>>> parent_coords = CascadedCoords(pos=[1, 0, 0])
>>> child_coords = CascadedCoords(pos=[1, 1, 0])
>>> parent_coords.assoc(child_coords, relative_coords='local')
>>> child_coords.worldpos()
array([2., 1., 0.])
>>> child_coords.translation
array([1., 1., 0.])
```

axis(ax)

changed()

Return False

This is used for CascadedCoords compatibility

Returns False – always return False

Return type bool

coords()

Return a deep copy of the Coordinates.

copy()

Return a deep copy of the Coordinates.

copy_coords()

Return a deep copy of the Coordinates.

copy_worldcoords()

Return a deep copy of the Coordinates.

difference_position(coords, translation_axis=True)

Return differences in positoin of given coords.

Parameters

- coords (skrobot.coordinates.Coordinates) given coordinates
- translation_axis (str or bool or None (optional)) we can take 'x', 'y', 'z', 'xy', 'yz', 'zx', 'xx', 'yy', 'zz', True or False(None).

Returns dif_pos – difference position of self coordinates and coords considering translation_axis.

Return type numpy.ndarray

Examples

difference_rotation(coords, rotation_axis=True)

Return differences in rotation of given coords.

Parameters

- coords (skrobot.coordinates.Coordinates) given coordinates
- rotation_axis(str or bool or None (optional)) we can take 'x', 'y', 'z', 'xx', 'yy', 'zz', 'xm', 'ym', 'zm', 'xy', 'yx', 'yz', 'zy', 'zx', 'xz', True or False(None).

Returns dif_rot – difference rotation of self coordinates and coords considering rotation_axis.

Return type numpy.ndarray

Examples

(continues on next page)

(continued from previous page)

Using mirror option ['xm', 'ym', 'zm'], you can allow differences of mirror direction.

disable_hook()

dissoc(child)

get_transform()

Return Transform object

Returns transform – corrensponding Transform to this coordinates

Return type skrobot.coordinates.base.Transform

inverse_rotate_vector(v)

inverse_transform_vector(v)

Transform vector in world coordinates to local coordinates

Parameters vec (numpy.ndarray) – 3d vector. We can take batch of vector like (batch_size, 3)

Returns transformed_point – transformed point

Return type numpy.ndarray

inverse_transformation(dest=None)

Return a invese transformation of this coordinate system.

Create a new coordinate with inverse transformation of this coordinate system.

$$\left(\begin{array}{cc} R^{-1} & -R^{-1}p \\ 0 & 1 \end{array}\right)$$

Parameters dest (*None or* skrobot.coordinates.Coordinates) – If dest is given, the result of transformation is in-placed to dest.

Returns dest – result of inverse transformation.

Return type *skrobot.coordinates.Coordinates*

move_coords(target_coords, local_coords)

Transform this coordinate so that local_coords to target_coords.

Parameters

- target_coords (skrobot.coordinates.Coordinates) target coords.
- local_coords (skrobot.coordinates.Coordinates) local coords to be aligned.

Returns self.worldcoords() – world coordinates.

Return type skrobot.coordinates.Coordinates

```
newcoords(c, pos=None, check_validity=True)
```

Update this coordinates.

This function records that this CascadedCoords has changed and recursively records the change to descendants of this CascadedCoords.

Parameters

- **c** (skrobot.coordinates.Coordinates *or numpy.ndarray*) If pos is *None*, *c* means new Coordinates. If pos is given, *c* means rotation matrix.
- pos (numpy.ndarray or None) new translation.
- **check_validity** (*bool*) If this value is *True*, check whether an input rotation and an input translation are valid.

orient_with_matrix(rotation_matrix, wrt='world')

Force update this coordinate system's rotation.

Parameters

- **rotation_matrix** (*numpy.ndarray*) 3x3 rotation matrix.
- wrt (str or skrobot.coordinates.Coordinates) reference coordinates.

```
parent_orientation(v, wrt)
```

```
parentcoords()
```

```
rotate(theta, axis, wrt='local')
```

Rotate this coordinate.

Rotate this coordinate relative to axis by theta radians with respect to wrt.

Parameters

```
• theta (float) - radian
```

```
• axis (str or numpy.ndarray) - 'x', 'y', 'z' or vector
```

• wrt (str or Coordinates) -

Return type self

rotate_vector(v)

Rotate 3-dimensional vector using rotation of this coordinate

```
Parameters v (numpy.ndarray) – vector shape of (3,)
```

Returns np.matmul(self.rotation, v) – rotated vector

Return type numpy.ndarray

```
>>> from skrobot.coordinates import Coordinates
>>> from numpy import pi
>>> c = Coordinates().rotate(pi, 'z')
>>> c.rotate_vector([1, 2, 3])
array([-1., -2., 3.])
```

rotate_with_matrix(matrix, wrt)

Rotate this coordinate by given rotation matrix.

This is a subroutine of self.rotate function.

Parameters

- mat (numpy.ndarray) rotation matrix shape of (3, 3)
- wrt (str or skrobot.coordinates.Coordinates) with respect to.

Returns self

Return type skrobot.coordinates.Coordinates

rpy_angle()

Return a pair of rpy angles of this coordinates.

```
Returns rpy_angle(self.rotation) – a pair of rpy angles. See also skrobot.coordinates.math.rpy_angle
```

Return type tuple(numpy.ndarray, numpy.ndarray)

Examples

```
>>> import numpy as np
>>> from skrobot.coordinates import Coordinates
>>> c = Coordinates().rotate(np.pi / 2.0, 'x').rotate(np.pi / 3.0, 'z')
>>> r.rpy_angle()
(array([ 3.84592537e-16, -1.04719755e+00, 1.57079633e+00]),
array([ 3.14159265, -2.0943951 , -1.57079633]))
```

transform(c, wrt='local', out=None)

Transform this coordinates

Parameters

- c (skrobot.coordinates.Coordinates) coordinates
- wrt (str or skrobot.coordinates.Coordinates) transform this coordinates with respect to wrt. If wrt is 'local' or self, multiply c from the right. If wrt is 'parent' or self.parent, transform c with respect to parentcoords. (multiply c from the left.) If wrt is Coordinates, transform c with respect to c.
- **out** (*None or* skrobot.coordinates.Coordinates) If the *out* is specified, set new coordinates to *out*. Note that if the *out* is given, these coordinates don't change.

Returns self – return self

Return type skrobot.coordinates.CascadedCoords

transform_vector(v)

"Return vector represented at world frame.

Vector v given in the local coords is converted to world representation.

Parameters v (numpy.ndarray) – 3d vector. We can take batch of vector like (batch_size, 3)

Returns transformed point – transformed point

Return type numpy.ndarray

```
transformation(c2, wrt='local')
```

```
translate(vec, wrt='local')
```

Translate this coordinates.

Note that this function changes this coordinates self. So if you don't want to change this class, use copy_worldcoords()

Parameters

- **vec** (*list or numpy.ndarray*) shape of (3,) translation vector. unit is [m] order.
- wrt (str or Coordinates (optional)) translate with respect to wrt.

Examples

```
>>> import numpy as np
>>> from skrobot.coordinates import Coordinates
>>> c = Coordinates()
>>> c.translation
array([0., 0., 0.], dtype=float32)
>>> c.translate([0.1, 0.2, 0.3])
>>> c.translation
array([0.1, 0.2, 0.3], dtype=float32)
```

```
>>> c = Coordinates()
>>> c.copy_worldcoords().translate([0.1, 0.2, 0.3])
>>> c.translation
array([0., 0., 0.], dtype=float32)
```

```
>>> c = Coordinates().rotate(np.pi / 2.0, 'y')
>>> c.translate([0.1, 0.2, 0.3])
>>> c.translation
array([ 0.3,  0.2, -0.1])
>>> c = Coordinates().rotate(np.pi / 2.0, 'y')
>>> c.translate([0.1, 0.2, 0.3], 'world')
>>> c.translation
array([0.1, 0.2, 0.3])
```

update(force=False)

worldcoords()

Calculate rotation and position in the world.

worldpos()

Return translation of this coordinate

See also skrobot.coordinates.Coordinates.translation

```
Returns self.translation – translation of this coordinate
         Return type numpy.ndarray
worldrot()
     Return rotation of this coordinate
     See also skrobot.coordinates.Coordinates.rotation
         Returns self.rotation – rotation matrix of this coordinate
         Return type numpy.ndarray
__eq__(value,/)
     Return self==value.
__ne__(value,/)
     Return self!=value.
__lt__(value,/)
     Return self<value.
le (value./)
     Return self<=value.
__gt__(value,/)
     Return self>value.
__ge__(value,/)
     Return self>=value.
__mul__(other_c)
     Return Transformed Coordinates.
     Note that this function creates new Coordinates and does not change translation and rotation, unlike trans-
     form function.
         Parameters other_c (skrobot.coordinates.Coordinates) - input coordinates.
         Returns out – transformed coordinates multiplied other_c from the right. T = T_{self}
             T_{\text{other}_c}.
         Return type skrobot.coordinates.Coordinates
__pow__(exponent)
     Return exponential homogeneous matrix.
     If exponent equals -1, return inverse transformation of this coords.
         Parameters exponent (numbers. Number) – exponent value. If exponent equals -1, return in-
             verse transformation of this coords. In current, support only -1 case.
         Returns out - output.
         Return type skrobot.coordinates.Coordinates
```

Attributes

descendants

dimension

Return dimension of this coordinate

Returns len(self.translation) – dimension of this coordinate

Return type int

dual_quaternion

Property of DualQuaternion

Return DualQuaternion representation of this coordinate.

Returns DualQuaternion - DualQuaternion representation of this coordinate

Return type skrobot.coordinates.dual_quaternion.DualQuaternion

name

Return this coordinate's name

Returns self. name – name of this coordinate

Return type str

parent

quaternion

Property of quaternion

Returns $\mathbf{q} - [\mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z}]$ quaternion

Return type numpy.ndarray

Examples

```
>>> from numpy import pi
>>> from skrobot.coordinates import make_coords
>>> c = make_coords()
>>> c.quaternion
array([1., 0., 0., 0.])
>>> c.rotate(pi / 3, 'y').rotate(pi / 5, 'z')
>>> c.quaternion
array([0.8236391 , 0.1545085 , 0.47552826, 0.26761657])
```

rotation

Return rotation matrix of this coordinates.

Returns self._rotation – 3x3 rotation matrix

Return type numpy.ndarray

translation

Return translation of this coordinates.

Return type numpy.ndarray

```
Returns self._translation – vector shape of (3, ). unit is [m]
```

Examples

```
>>> from skrobot.coordinates import Coordinates
>>> c = Coordinates()
>>> c.translation
array([0., 0., 0.])
>>> c.translate([0.1, 0.2, 0.3])
>>> c.translation
array([0.1, 0.2, 0.3])
```

x_axis

Return x axis vector of this coordinates.

```
Returns axis – x axis.
```

Return type numpy.ndarray

y_axis

Return y axis vector of this coordinates.

```
Returns axis – y axis.
```

Return type numpy.ndarray

z_axis

Return z axis vector of this coordinates.

```
Returns axis – z axis.
```

Return type numpy.ndarray

3.1.2 Coordinates utilities

skrobot.coordinates.geo.midcoords	Returns mid (or p) coordinates of given two coordinates c1 and c2.
skrobot.coordinates.geo.	Orient axis to the direction
orient_coords_to_axis	
skrobot.coordinates.base.random_coords	Return Coordinates class has random translation and ro-
	tation
skrobot.coordinates.base.	
coordinates_distance	
skrobot.coordinates.base.transform_coords	Return Coordinates by applying c1 to c2 from the left

skrobot.coordinates.geo.midcoords

skrobot.coordinates.geo.midcoords(p, c1, c2)

Returns mid (or p) coordinates of given two coordinates c1 and c2.

Parameters

- **p** (*float*) ratio of c1:c2
- c1 (skrobot.coordinates.Coordinates) Coordinates
- c2 (skrobot.coordinates.Coordinates) Coordinates

Returns coordinates – midcoords

Return type skrobot.coordinates.Coordinates

Examples

```
>>> from skrobot.coordinates import Coordinates
>>> from skrobot.coordinates.geo import midcoords
>>> c1 = Coordinates()
>>> c2 = Coordinates(pos=[0.1, 0, 0])
>>> c = midcoords(0.5, c1, c2)
>>> c.translation
array([0.05, 0. , 0. ])
```

skrobot.coordinates.geo.orient_coords_to_axis

skrobot.coordinates.geo.orient_coords_to_axis(target_coords, v, axis='z', eps=0.005)

Orient axis to the direction

Orient axis in target_coords to the direction specified by v.

Parameters

- target_coords (skrobot.coordinates.Coordinates) -
- v (list or numpy.ndarray) position of target [x, y, z]
- axis(list or string or numpy.ndarray) see _wrap_axis function
- eps (float (optional)) eps

Returns target_coords

Return type skrobot.coordinates.Coordinates

Examples

```
>>> import numpy as np
>>> from skrobot.coordinates import Coordinates
>>> from skrobot.coordinates.geo import orient_coords_to_axis
>>> c = Coordinates()
>>> oriented_coords = orient_coords_to_axis(c, [1, 0, 0])
>>> oriented_coords.translation
array([0., 0., 0.])
>>> oriented_coords.rpy_angle()
(array([0., 1.57079633, 0.]),
array([3.14159265, 1.57079633, 3.14159265]))
```

```
>>> c = Coordinates(pos=[0, 1, 0]).rotate(np.pi / 3, 'y')
>>> oriented_coords = orient_coords_to_axis(c, [0, 1, 0])
>>> oriented_coords.translation
array([0., 1., 0.])
>>> oriented_coords.rpy_angle()
(array([-5.15256299e-17, 1.04719755e+00, -1.57079633e+00]),
array([3.14159265, 2.0943951 , 1.57079633]))
```

skrobot.coordinates.base.random coords

```
skrobot.coordinates.base.random_coords()
```

Return Coordinates class has random translation and rotation

skrobot.coordinates.base.coordinates distance

skrobot.coordinates.base.coordinates_distance(c1, c2, c=None)

skrobot.coordinates.base.transform coords

```
skrobot.coordinates.base.transform_coords(c1, c2, out=None)
Return Coordinates by applying c1 to c2 from the left
```

Parameters

- c1 (skrobot.coordinates.Coordinates) -
- c2 (skrobot.coordinates.Coordinates) Coordinates
- **c3** (skrobot.coordinates.Coordinates *or None*) Output argument. If this value is specified, the results will be in-placed.

Returns Coordinates(pos=translation, rot=q) – new coordinates

Return type *skrobot.coordinates.Coordinates*

Examples

```
>>> from skrobot.coordinates import Coordinates
>>> from skrobot.coordinates import transform_coords
>>> from numpy import pi
>>> c1 = Coordinates()
>>> c2 = Coordinates()
>>> c3 = transform_coords(c1, c2)
>>> c3.translation
array([0., 0., 0.])
>>> c3.rotation
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]
>>> c1 = Coordinates().translate([0.1, 0.2, 0.3]).rotate(pi / 3.0, 'x')
>>> c2 = Coordinates().translate([0.3, -0.3, 0.1]).rotate(pi / 2.0, 'y')
>>> c3 = transform_coords(c1, c2)
>>> c3.translation
array([ 0.4
                  , -0.03660254, 0.09019238])
>>> c3.rotation
>>> c3.rotation
array([[ 1.94289029e-16, 0.00000000e+00, 1.00000000e+00],
       [8.66025404e-01, 5.00000000e-01, -1.66533454e-16],
       [-5.00000000e-01, 8.66025404e-01, 2.77555756e-17]])
```

3.1.3 Quaternion Classes

skrobot.coordinates.quaternion.Quaternion	Class for handling Quaternion.
skrobot.coordinates.dual_quaternion.	Class for handling dual quaternions and their interpola-
DualQuaternion	tions.

skrobot.coordinates.quaternion.Quaternion

class skrobot.coordinates.quaternion.**Quaternion**(w=1.0, x=0.0, y=0.0, z=0.0, q=None) Class for handling Quaternion.

Parameters

```
w(float or numpy.ndarray) -
x(float) -
y(float) -
z(float) -
q(None or numpy.ndarray) - if q is not specified, use w, x, y, z.
```

Examples

```
>>> from skrobot.coordinates.quaternion import Quaternion
>>> q = Quaternion()
>>> q
#<Quaternion 0x1283bde48 w: 1.0 x: 0.0 y: 0.0 z: 0.0>
>>> q = Quaternion([1, 2, 3, 4])
>>> q
#<Quaternion 0x1283cad68 w: 1.0 x: 2.0 y: 3.0 z: 4.0>
>>> q = Quaternion(q=[1, 2, 3, 4])
>>> q
#<Quaternion 0x1283bd438 w: 1.0 x: 2.0 y: 3.0 z: 4.0>
>>> q = Quaternion(1, 2, 3, 4)
>>> q
#<Quaternion 0x1283bd438 w: 1.0 x: 2.0 y: 3.0 z: 4.0>
>>> q = Quaternion(w=0.0, x=1.0, y=0.0, z=0.0)
>>> q
#<Quaternion 0x128400198 w: 1.0 x: 2.0 y: 3.0 z: 4.0>
>>> q = Quaternion(w=0.0, x=1.0, y=0.0, z=0.0)
>>> q
#<Quaternion 0x1283cc2e8 w: 0.0 x: 1.0 y: 0.0 z: 0.0>
```

Methods

T()

Return 4x4 homogeneous transformation matrix.

Returns matrix – homogeneous transformation matrix shape of (4, 4)

Return type numpy.ndarray

Examples

(continues on next page)

(continued from previous page)

copy()

Return copy of this Quaternion

Returns Quaternion(q=self.q.copy()) – copy of this quaternion

Return type skrobot.coordinates.quaternion.Quaternion

normalize()

Normalize this quaternion.

Note that this function changes wxyz property.

Examples

```
>>> from skrobot.coordinates.quaternion import Quaternion
    >>> q = Quaternion([1, 2, 3, 4])
    >>> q.q
    array([1., 2., 3., 4.])
    >>> q.normalize()
    >>> q.q
    array([0.18257419, 0.36514837, 0.54772256, 0.73029674])
__eq__(value,/)
    Return self==value.
__ne__(value,/)
    Return self!=value.
__lt__(value,/)
    Return self<value.
__le__(value,/)
    Return self<=value.
__gt__(value,/)
    Return self>value.
__ge__(value,/)
    Return self>=value.
__neg__()
__add__(cls)
__sub__(cls)
__mul__(cls)
__rmul__(cls)
__div__(cls)
__truediv__(cls)
```

Attributes

angle

Return rotation angle of this quaternion

Returns theta – rotation angle with respect to self.axis

Return type float

axis

Return axis of this quaternion.

Note that this property return normalized axis.

Returns axis - normalized axis vector

Return type numpy.ndarray

conjugate

Return conjugate of this quaternion

Returns Quaternion – new Quaternion class has this quaternion's conjugate

Return type skrobot.coordinates.quaternion.Quaternion

Examples

```
>>> from skrobot.coordinates.quaternion import Quaternion

>>> q = Quaternion()

>>> q.conjugate

#<Quaternion 0x12f2dfb38 w: 1.0 x: -0.0 y: -0.0 z: -0.0>

>>> q.q = [0, 1, 0, 0]

>>> q.conjugate

#<Quaternion 0x12f303c88 w: 0.0 x: -1.0 y: -0.0 z: -0.0>
```

inverse

Return inverse of this quaternion

Returns q – new Quaternion class has inverse of this quaternion

Return type *skrobot.coordinates.quaternion.Quaternion*

Examples

```
>>> from skrobot.coordinates.quaternion import Quaternion
>>> q = Quaternion()
>>> q
#<Quaternion 0x127e6da58 w: 1.0 x: 0.0 y: 0.0 z: 0.0>
>>> q.inverse
#<Quaternion 0x1281bbda0 w: 1.0 x: -0.0 y: -0.0 z: -0.0>
>>> q.q = [0, 1, 0, 0]
>>> q.inverse
#<Quaternion 0x1282b0cc0 w: 0.0 x: -1.0 y: -0.0 z: -0.0>
```

norm

Return norm of this quaternion

Returns quaternion_norm(self.q) – norm of this quaternion

Return type float

Examples

normalized

Return Normalized quaternion.

Returns normalized quaternion – return quaternion which is norm == 1.0.

Return type *skrobot.coordinates.quaternion.Quaternion*

Examples

```
>>> from skrobot.coordinates.quaternion import Quaternion
>>> q = Quaternion([1, 2, 3, 4])
>>> normalized_q = q.normalized
>>> normalized_q.q
array([0.18257419, 0.36514837, 0.54772256, 0.73029674])
>>> q.q
array([1., 2., 3., 4.])
```

q

Return quaternion

Returns self._ \mathbf{q} – [w, x, y, z] quaternion

Return type numpy.ndarray

Examples

```
>>> from skrobot.coordinates.quaternion import Quaternion
>>> q = Quaternion()
>>> q.q
array([1., 0., 0., 0.])
>>> q = Quaternion(w=0.0, x=1.0, y=0.0, z=0.0)
>>> q.q
array([0., 1., 0., 0.])
```

rotation

Return rotation matrix.

Note that this property internally normalizes quaternion.

Returns quaternion2matrix(self.q) – 3x3 rotation matrix

3.1. Coordinates 33

Return type numpy.ndarray

Examples

```
>>> from skrobot.coordinates.quaternion import Quaternion
>>> q = Quaternion()
>>> q
#<Quaternion 0x12f1aa6a0 w: 1.0 x: 0.0 y: 0.0 z: 0.0>
>>> q.rotation
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
>>> q.q = [0, 1, 0, 0]
>>> q
#<Quaternion 0x12f1aa6a0 w: 0.0 x: 1.0 y: 0.0 z: 0.0>
>>> q.rotation
array([[ 1., 0., 0.],
       [0., -1., 0.],
       [0., 0., -1.]
\Rightarrow q.q = [1, 2, 3, 4]
>>> q
#<Quaternion 0x12f1aa6a0 w: 1.0 x: 2.0 y: 3.0 z: 4.0>
>>> q.rotation
array([[-0.66666667, 0.13333333, 0.73333333],
       [0.66666667, -0.33333333, 0.66666667],
       [0.33333333, 0.93333333, 0.13333333]])
Return w element
   Returns self.q[0] – w element of this quaternion
   Return type float
```

W

X

Return x element

Returns self.q[1] - x element of this quaternion

Return type float

xyz

Return xyz vector of this quaternion

Returns quaternion_xyz – xyz elements of this quaternion

Return type numpy.ndarray

у

Return y element

Returns self.q[2] – y element of this quaternion

Return type float

Z

Return z element

Returns self.q[3] – z element of this quaternion

Return type float

skrobot.coordinates.dual_quaternion.DualQuaternion

```
class skrobot.coordinates.dual_quaternion.DualQuaternion(qr=[1, 0, 0, 0], qd=[0, 0, 0, 0], enforce\_unit\_norm=False)
```

Class for handling dual quaternions and their interpolations.

Parameters

```
• qr(list or numpy.ndarray)—
```

- qd (list or numpy.ndarray) element of dual quaternion
- enforce_unit_norm (bool (optional)) if True, norm should be 1.0.

Methods

T()

Return 4x4 homogeneous transformation matrix.

Returns matrix – homogeneous transformation matrix shape of (4, 4)

Return type numpy.ndarray

Examples

```
>>> from numpy import pi
>>> from skrobot.coordinates import Coordinates
>>> from skrobot.coordinates.dual_quaternion import DualQuaternion
>>> dq = DualQuaternion()
>>> dq.T()
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]]
>>> dq = Coordinates().rotate(pi / 2.0, 'y').
\hookrightarrow translate((0.1, 0.2, 0.3)).
                                                              dual_quaternion
>>> dq.T()
array([[ 2.22044605e-16, 0.00000000e+00,
                                            1.00000000e+00,
         3.00000000e-01],
       [ 0.00000000e+00, 1.00000000e+00,
                                            0.00000000e+00,
         2.00000000e-01],
       [-1.000000000e+00, 0.00000000e+00,
                                            2.22044605e-16,
        -1.00000000e-01],
       [ 0.00000000e+00, 0.00000000e+00,
                                            0.00000000e+00,
         1.00000000e+00]])
```

copy()

Return a copy of this quaternion.

Returns copied DualQuaternion instance

Return type DualQuaternion

3.1. Coordinates 35

```
difference_position(other_dq)
     Return difference position
         Parameters other_dq (skrobot.coordinates.dual_quaternion.DualQuaternion) -
             dual quaternion
         Returns dif_pos – difference position's norm
         Return type float
difference_rotation(other_dq)
     Return difference rotation distance
         Parameters other_dq (skrobot.coordinates.dual_quaternion.DualQuaternion) -
             dual quaternion
         Returns dif_rot – angle distance in radian.
         Return type float
enforce_positive_q_rot_w()
static interpolate (dq0, dq1, t)
     Return interpolated dual quaternion
         Parameters
             • dq0 (skrobot.coordinates.dual_quaternion.DualQuaternion) -
             • dq1 (skrobot.coordinates.dual_quaternion.DualQuaternion) - dual quaternion
             • t (float) – ratio of interpolation. Must be 0 \le t \le 1.0.
normalize()
     Normalize this dual quaternion
     Note that this function changes property.
         Returns self – return self
         \textbf{Return type} \ \textit{skrobot.coordinates.dual\_quaternion.DualQuaternion}
pose()
     Return [x, y, z, wx, wy, wz, wq] elements.
         Returns pose – [x, y, z, wx, wy, wz, wq] pose
         Return type numpy.ndarray
screw_axis()
     Return screw axis
     Calculates rotation, translation and screw axis from dual quaternion.
         Returns screw_axis, theta, translation – screw axis of this dual quaternion. rotation angle in
             radian. translation
         Return type tuple(numpy.ndarray, float, float)
__eq__(value,/)
     Return self==value.
__ne__(value,/)
     Return self!=value.
__lt__(value,/)
     Return self<value.
```

```
__le__(value,/)
    Return self<=value.
__gt__(value,/)
    Return self>value.
__ge__(value,/)
    Return self>=value.
__add__(val)
__mul__(val)
__rmul__(val)
```

Attributes

angle

Return rotation angle of this dual quaternion

Returns self.qr.angle – this dual quaternion's rotation angle with respect to self.axis. See skrobot.coordinates.quaternion.Quaternion.angle.

Return type float

axis

Return axis of this dual quaternion

Returns self.qr.axis – this dual quaternion's axis. See See skrobot.coordinates.quaternion.Quaternion.axis.

Return type numpy.ndarray

conjugate

Return conjugate of this dual quaternion

Returns DualQuaternion – new DualQuaternion class has this dual quaternion's conjugate

Return type *skrobot.coordinates.dual_quaternion.DualQuaternion*

dq

Return flatten vector of this dual quaternion

Returns np.concatenate([self.qr.q, self.qd.q]) – (1x8) vector of this dual quaternion

Return type numpy.ndarray

Examples

```
>>> from skrobot.coordinates.dual_quaternion import DualQuaternion
>>> dq = DualQuaternion()
>>> dq.dq
array([1., 0., 0., 0., 0., 0., 0.])
```

inverse

Return inverse of this dual quaternion

Returns dq – new DualQuaternion class has inverse of this dual quaternion

Return type skrobot.coordinates.dual_quaternion.DualQuaternion

3.1. Coordinates 37

norm

Return pair of norm of this dual quaternion

```
Returns qr_norm, qd_norm – qr and qd's norm

Return type tuple(float, float)
```

Examples

```
>>> from skrobot.coordinates.dual_quaternion import DualQuaternion
>>> dq = DualQuaternion()
>>> dq.norm
(1.0, 0.0)
```

normalized

Return normalized this dual quaternion

Returns skrobot.coordinates.dual_quaternion.DualQuaternion normalized dual quaternion

Return type *DualQuaternion*(qr, qd, True)

qd

Return translation quaternion

Returns self._qd – quaternion indicating translation

Return type skrobot.coordinates.quaternion.Quaternion

qr

Return orientation

```
Returns self._qr - [w, x, y, z] order
```

Return type numpy.narray

quaternion

Return this dual quaternion's qr (rotation)

Returns dq.qr - rotation quaternion

Return type skrobot.coordinates.quaternion.Quaternion

rotation

Return rotation matrix of this dual quaternion

Returns dq.qr.rotation – 3x3 rotation matrix

Return type numpy.ndarray

Examples

(continues on next page)

(continued from previous page)

scalar

The scalar part of the dual quaternion.

```
\boldsymbol{Returns} \hspace{0.2cm} \boldsymbol{scalar} - scalar
```

Return type float

translation

Return translation of this dual quaternion.

```
Returns q_translation.xyz – vector shape of (3, ). unit is [m]
```

Return type numpy.ndarray

Examples

```
>>> from skrobot.coordinates import Coordinates
>>> c = Coordinates()
>>> c.dual_quaternion.translation
array([0., 0., 0.])
>>> c.translate([0.1, 0.2, 0.3])
>>> c.dual_quaternion.translation
array([0.1, 0.2, 0.3])
```

3.2 Models

3.2.1 Robot Model class

skrobot.model.RobotModel

skrobot.model.RobotModel

class skrobot.model.RobotModel(link_list=None, joint_list=None, root_link=None)

Methods

T()

Return 4x4 homogeneous transformation matrix.

Returns matrix – homogeneous transformation matrix shape of (4, 4)

Return type numpy.ndarray

Examples

```
>>> from numpy import pi
>>> from skrobot.coordinates import make_coords
>>> c = make_coords()
>>> c.T()
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]
>>> c.translate([0.1, 0.2, 0.3])
>>> c.rotate(pi / 2.0, 'y')
array([[ 2.22044605e-16, 0.00000000e+00,
                                           1.00000000e+00,
         1.00000000e-01],
       [ 0.00000000e+00, 1.00000000e+00,
                                           0.00000000e+00.
         2.00000000e-01],
       [-1.000000000e+00, 0.00000000e+00,
                                           2.22044605e-16,
         3.00000000e-01],
       [ 0.00000000e+00, 0.0000000e+00,
                                           0.00000000e+00,
         1.00000000e+00]])
```

angle_vector(av=None, return_av=None)

Returns angle vector

If av is given, it updates angles of all joint. If given av violate min/max range, the value is modified.

```
assoc(child, relative coords='world', force=False, **kwargs)
```

Associate child coords to this coordinate system.

If *relative_coords* is *None* or 'world', the translation and rotation of childcoords in the world coordinate system do not change. If *relative_coords* is specified, childcoords is assoced at translation and rotation of *relative_coords*. By default, if child is already assoced to some other coords, raise an exception. But if *force* is *True*, you can overwrite the existing assoc relation.

Parameters

- child (CascadedCoords) child coordinates.
- relative_coords (None or Coordinates or str) child coordinates' relative coordinates.
- **force** (*bool*) predicate for overwriting the existing assoc-relation

Returns child – assoced child.

Return type CascadedCoords

```
>>> from skrobot.coordinates import CascadedCoords
>>> parent_coords = CascadedCoords(pos=[1, 0, 0])
>>> child_coords = CascadedCoords(pos=[1, 1, 0])
>>> parent_coords.assoc(child_coords)
#<CascadedCoords 0x7f1d30e29510 1.000 1.000 0.000 / 0.0 -0.0 0.0>
>>> child_coords.worldpos()
array([1., 1., 0.])
>>> child_coords.translation
array([0., 1., 0.])
```

None and 'world' have the same meaning.

```
>>> parent_coords = CascadedCoords(pos=[1, 0, 0])
>>> child_coords = CascadedCoords(pos=[1, 1, 0])
>>> parent_coords.assoc(child_coords, relative_coords='world')
#<CascadedCoords 0x7f1d30e29510 1.000 1.000 0.000 / 0.0 -0.0 0.0>
>>> child_coords.worldpos()
array([1., 1., 0.])
```

If *relative_coords* is 'local', *child* is associated at world translation and world rotation of *child* from this coordinate system.

```
>>> parent_coords = CascadedCoords(pos=[1, 0, 0])
>>> child_coords = CascadedCoords(pos=[1, 1, 0])
>>> parent_coords.assoc(child_coords, relative_coords='local')
>>> child_coords.worldpos()
array([2., 1., 0.])
>>> child_coords.translation
array([1., 1., 0.])
```

axis(ax)

```
\verb|calc_inverse_kinematics_nspace_from_link_list(|link_list, avoid_nspace\_gain=0.01, |link_list|)|
```

union_link_list=None, n_joint_dimension=None, null_space=None, additional_nspace_list=None, weight=None)

Calculate all weight from link list.

```
calc_jacobian_for_interlocking_joints(link_list, interlocking_joint_pairs=None)
```

calc_jacobian_from_link_list(move_target, link_list=None, transform_coords=None,

rotation_axis=None, translation_axis=None, col_offset=0, dim=None, jacobian=None, additional_jacobi_dimension=0, n_joint_dimension=None, *args, **kwargs)

calc_joint_angle_from_min_max_table(index, j, av)

 $\label{lem:calc_joint_angle_speed} \textbf{\textit{calc_joint_angle_speed}} \\ \textbf{\textit{union_vel}, \textit{angle_speed=None}, \textit{angle_speed_blending=0.5}, \textit{jacobi=None}, \\ \textbf{\textit{j_sharp=None}, \textit{null_space=None}, *args, **kwargs)} \\ \\$

```
calc_joint_angle_speed_gain(union_link_list, dav, periodic_time)
calc_nspace_from_joint_limit(avoid_nspace_gain, union_link_list, weight)
     Calculate null-space according to joint limit.
calc_target_axis_dimension(rotation_axis, translation_axis)
     rotation-axis, translation-axis -> both list and atom OK.
calc_target_joint_dimension(link_list)
calc_union_link_list(link_list)
calc_vel_for_interlocking_joints(link_list, interlocking_joint_pairs=None)
     Calculate 0 velocity for keeping interlocking joint.
     at the same joint angle.
calc_vel_from_pos(dif_pos, translation_axis, p_limit=100.0)
     Calculate velocity from difference position
         Parameters
             • dif_pos (np.ndarray) - [m] order
             • translation_axis (str) - see calc dif with axis
         Returns vel_p
         Return type np.ndarray
calc_vel_from_rot(dif rot, rotation axis, r limit=0.5)
calc_weight_from_joint_limit(avoid_weight_gain, link_list, union_link_list, weight,
                                   n_joint_dimension=None)
     Calculate weight according to joint limit.
changed()
     Return False
     This is used for CascadedCoords compatibility
         Returns False – always return False
         Return type bool
collision_avoidance_link_pair_from_link_list(link_list, obstacles=None)
compute_qp_common(target_coords, move_target, dt, link_list=None, gain=0.85, weight=1.0,
                     translation_axis=True, rotation_axis=True, dof_limit_gain=0.5)
compute_velocity(target coords, move target, dt, link list=None, gain=0.85, weight=1.0,
                    translation axis=True, rotation axis=True, dof limit gain=0.5, fast=True,
                    sym proj=False, solver='cvxopt', *args, **kwargs)
coords()
     Return a deep copy of the Coordinates.
copy()
     Return a deep copy of the Coordinates.
copy_coords()
     Return a deep copy of the Coordinates.
copy_worldcoords()
     Return a deep copy of the Coordinates.
```

difference_position(coords, translation axis=True)

Return differences in position of given coords.

Parameters

- coords (skrobot.coordinates.Coordinates) given coordinates
- translation_axis (str or bool or None (optional)) we can take 'x', 'y', 'z', 'xy', 'yz', 'zx', 'xx', 'yy', 'zz', True or False(None).

Returns dif_pos – difference position of self coordinates and coords considering translation_axis.

Return type numpy.ndarray

Examples

difference_rotation(coords, rotation axis=True)

Return differences in rotation of given coords.

Parameters

- coords (skrobot.coordinates.Coordinates) given coordinates
- rotation_axis(str or bool or None (optional)) we can take 'x', 'y', 'z', 'xx', 'yy', 'zz', 'xm', 'ym', 'zm', 'xy', 'yx', 'yz', 'zy', 'zx', 'xz', True or False(None).

Returns dif_rot – difference rotation of self coordinates and coords considering rotation_axis.

Return type numpy.ndarray

Examples

```
>>> from numpy import pi
>>> from skrobot.coordinates import Coordinates
>>> from skrobot.coordinates.math import rpy_matrix
>>> coord1 = Coordinates()
>>> coord2 = Coordinates(rot=rpy_matrix(pi / 2.0, pi / 3.0, pi / 5.0))
>>> coord1.difference_rotation(coord2)
array([-0.32855112, 1.17434985, 1.05738936])
>>> coord1.difference_rotation(coord2, rotation_axis=False)
```

(continues on next page)

(continued from previous page)

Using mirror option ['xm', 'ym', 'zm'], you can allow differences of mirror direction.

disable_hook()

dissoc(child)

find_joint_angle_limit_weight_from_union_link_list(union_link_list)

find_link_path(src_link, target_link)

Find paths of src_link to target_link

Parameters

- **src_link** (*skrobot.model.link.Link*) source link.
- target_link (skrobot.model.link.Link) target link.

Returns ret – If the links are connected, return Link list. Otherwise, return an empty list.

Return type List[skrobot.model.link.Link]

find_link_route(to, frm=None)

fix_leg_to_coords(fix_coords, leg='both', mid=0.5)

Fix robot's legs to a coords

In the Following codes, leged robot is assumed.

Parameters

- **fix_coords** (Coordinates) target coordinate
- leg (string) ['both', 'rleg', 'rleg', 'left', 'right']
- mid (float) ratio of legs coord.

get_transform()

Return Transform object

Returns transform – corrensponding Transform to this coordinates

Return type skrobot.coordinates.base.Transform

ik_convergence_check(dif_pos, dif_rot, rotation_axis, translation_axis, thre, rthre, centroid_thre=None, target_centroid_pos=None, centroid_offset_func=None, cog_translation_axis=None, update_mass_properties=True)

check ik convergence.

Parameters

- dif_pos (list of np.ndarray) -
- dif_rot (list of np.ndarray) -
- translation_axis (list of axis) -
- rotation_axis (list of axis) see _wrap_axis

init_pose()

inverse_kinematics(target_coords, move_target=None, link_list=None, **kwargs)

Solve inverse kinematics.

solve inverse kinematics, move move-target to target-coords look-at- target suppots t, nil, float-vector, coords, list of float-vector, list of coords link-list is set by default based on move-target -> root link link-list.

inverse_kinematics_loop(dif_pos, dif_rot, move_target, link_list=None, target_coords=None, **kwargs)
move move_target using dif_pos and dif_rot.

Parameters TODO -

Return type TODO

Solve look at inverse kinematics

Parameters look_at(list or np.ndarray or Coordinates) -

inverse_kinematics_optimization(target_coords, move_target=None, link_list=None,

regularization_parameter=None, init_angle_vector=None, translation_axis=True, rotation_axis=True, stop=100, dt=0.005, inverse_kinematics_hook=[], thre=0.001, rthre=0.017453292519943295, *args, **kwargs)

inverse_rotate_vector(v)

inverse_transform_vector(v)

Transform vector in world coordinates to local coordinates

Parameters vec (numpy.ndarray) – 3d vector. We can take batch of vector like (batch_size, 3)

Returns transformed_point – transformed point

Return type numpy.ndarray

inverse_transformation(dest=None)

Return a invese transformation of this coordinate system.

Create a new coordinate with inverse transformation of this coordinate system.

$$\left(\begin{array}{cc} R^{-1} & -R^{-1}p \\ 0 & 1 \end{array}\right)$$

Parameters dest (*None or* skrobot.coordinates.Coordinates) – If dest is given, the result of transformation is in-placed to dest.

```
Returns dest – result of inverse transformation.
```

Return type *skrobot.coordinates.Coordinates*

```
link_lists(to, frm=None)
```

Find link list from to link to frm link.

load_urdf(urdf)

load_urdf_file(file_obj)

look_at_hand(coords)

move_coords(target_coords, local_coords)

Transform this coordinate so that local_coords to target_coords.

Parameters

- target_coords (skrobot.coordinates.Coordinates) target coords.
- local_coords (skrobot.coordinates.Coordinates) local coords to be aligned.

Returns self.worldcoords() - world coordinates.

Return type *skrobot.coordinates.Coordinates*

```
move_end_pos(pos, wrt='local', *args, **kwargs)
```

move_end_rot(angle, axis, wrt='local', *args, **kwargs)

move_joints(union_vel, union_link_list=None, periodic_time=0.05, joint_args=None, move_joints_hook=None, *args, **kwargs)

obstacles=None, *args, **kwargs)

move_joints_avoidance(union_vel, union_link_list=None, link_list=None, n_joint_dimension=None, weight=None, null_space=None, avoid_nspace_gain=0.01, avoid_weight_gain=1.0, avoid_collision_distance=200, avoid_collision_null_gain=1.0, avoid_collision_joint_gain=1.0, collision_avoidance_link_pair=None, cog_gain=0.0, target_centroid_pos=None, centroid_offset_func=None, cog_translation_axis='z', cog_null_space=False, additional_weight_list=None, additional_nspace_list=None, jacobi=None,

newcoords(*c*, *pos=None*, *check_validity=True*)

Update this coordinates.

This function records that this CascadedCoords has changed and recursively records the change to descendants of this CascadedCoords.

Parameters

- **c** (skrobot.coordinates.Coordinates *or numpy.ndarray*) If pos is *None*, *c* means new Coordinates. If pos is given, *c* means rotation matrix.
- pos (numpy.ndarray or None) new translation.
- **check_validity** (*bool*) If this value is *True*, check whether an input rotation and an input translation are valid.

orient_with_matrix(rotation_matrix, wrt='world')

Force update this coordinate system's rotation.

Parameters

- **rotation_matrix** (*numpy.ndarray*) 3x3 rotation matrix.
- wrt(str or skrobot.coordinates.Coordinates) reference coordinates.

```
parent_orientation(v, wrt)
parentcoords()
reset_joint_angle_limit_weight(union_link_list)
reset_manip_pose()
reset_pose()
rotate(theta, axis, wrt='local')
     Rotate this coordinate.
     Rotate this coordinate relative to axis by theta radians with respect to wrt.
         Parameters
             • theta (float) - radian
             • axis (str or numpy.ndarray) - 'x', 'y', 'z' or vector
             • wrt(str or Coordinates)-
         Return type self
rotate_vector(v)
     Rotate 3-dimensional vector using rotation of this coordinate
         Parameters v (numpy.ndarray) – vector shape of (3,)
         Returns np.matmul(self.rotation, v) – rotated vector
         Return type numpy.ndarray
     Examples
     >>> from skrobot.coordinates import Coordinates
     >>> from numpy import pi
     >>> c = Coordinates().rotate(pi, 'z')
     >>> c.rotate_vector([1, 2, 3])
     array([-1., -2., 3.])
rotate_with_matrix(matrix, wrt)
     Rotate this coordinate by given rotation matrix.
     This is a subroutine of self.rotate function.
         Parameters
             • mat (numpy.ndarray) – rotation matrix shape of (3, 3)
             • wrt (str or skrobot.coordinates.Coordinates) - with respect to.
         Returns self
         Return type skrobot.coordinates.Coordinates
rpy_angle()
     Return a pair of rpy angles of this coordinates.
         Returns rpy_angle(self.rotation)
                                                    pair
                                                           of
                                                                rpy
                                                                      angles.
                                                                                      See
                                                                                            also
             skrobot.coordinates.math.rpy_angle
         Return type tuple(numpy.ndarray, numpy.ndarray)
```

```
>>> import numpy as np
>>> from skrobot.coordinates import Coordinates
>>> c = Coordinates().rotate(np.pi / 2.0, 'x').rotate(np.pi / 3.0, 'z')
>>> r.rpy_angle()
(array([ 3.84592537e-16, -1.04719755e+00, 1.57079633e+00]),
array([ 3.14159265, -2.0943951 , -1.57079633]))
```

self_collision_check()

Return collision link pair

Returns

- is_collision (bool) True if a collision occurred between any pair of objects and False otherwise
- names (*set of 2-tuple*) The set of pairwise collisions. Each tuple contains two names in alphabetical order indicating that the two corresponding objects are in collision.

transform(c, wrt='local', out=None)

Transform this coordinates

Parameters

- c (skrobot.coordinates.Coordinates) coordinates
- wrt (str or skrobot.coordinates.Coordinates) transform this coordinates with respect to wrt. If wrt is 'local' or self, multiply c from the right. If wrt is 'parent' or self.parent, transform c with respect to parentcoords. (multiply c from the left.) If wrt is Coordinates, transform c with respect to c.
- **out** (*None or* skrobot.coordinates.Coordinates) If the *out* is specified, set new coordinates to *out*. Note that if the *out* is given, these coordinates don't change.

Returns self – return self

Return type skrobot.coordinates.CascadedCoords

transform_vector(v)

"Return vector represented at world frame.

Vector v given in the local coords is converted to world representation.

Parameters v (numpy.ndarray) – 3d vector. We can take batch of vector like (batch_size, 3)

Returns transformed_point - transformed point

Return type numpy.ndarray

transformation(c2, wrt='local')

```
translate(vec, wrt='local')
```

Translate this coordinates.

Note that this function changes this coordinates self. So if you don't want to change this class, use copy_worldcoords()

Parameters

- **vec** (list or numpy.ndarray) shape of (3,) translation vector. unit is [m] order.
- wrt (str or Coordinates (optional)) translate with respect to wrt.

__le__(value,/)

Return self<=value.

```
>>> import numpy as np
    >>> from skrobot.coordinates import Coordinates
    >>> c = Coordinates()
    >>> c.translation
    array([0., 0., 0.], dtype=float32)
    >>> c.translate([0.1, 0.2, 0.3])
    >>> c.translation
    array([0.1, 0.2, 0.3], dtype=float32)
    >>> c = Coordinates()
    >>> c.copy_worldcoords().translate([0.1, 0.2, 0.3])
    >>> c.translation
    array([0., 0., 0.], dtype=float32)
    >>> c = Coordinates().rotate(np.pi / 2.0, 'y')
    >>> c.translate([0.1, 0.2, 0.3])
    >>> c.translation
    array([ 0.3, 0.2, -0.1])
    >>> c = Coordinates().rotate(np.pi / 2.0, 'y')
    >>> c.translate([0.1, 0.2, 0.3], 'world')
    >>> c.translation
    array([0.1, 0.2, 0.3])
update(force=False)
worldcoords()
    Calculate rotation and position in the world.
worldpos()
    Return translation of this coordinate
    See also skrobot.coordinates.Coordinates.translation
        Returns self.translation – translation of this coordinate
        Return type numpy.ndarray
worldrot()
    Return rotation of this coordinate
    See also skrobot.coordinates.Coordinates.rotation
        Returns self.rotation – rotation matrix of this coordinate
        Return type numpy.ndarray
__eq__(value,/)
    Return self==value.
__ne__(value,/)
    Return self!=value.
__lt__(value,/)
    Return self<value.
```

```
__gt__(value,/)
     Return self>value.
__ge__(value,/)
     Return self>=value.
__mul__(other c)
     Return Transformed Coordinates.
     Note that this function creates new Coordinates and does not change translation and rotation, unlike trans-
     form function.
         Parameters other_c (skrobot.coordinates.Coordinates) - input coordinates.
         Returns out – transformed coordinates multiplied other_c from the right. T = T_{self}
             T_{other_c}.
         Return type skrobot.coordinates.Coordinates
__pow__(exponent)
     Return exponential homogeneous matrix.
     If exponent equals -1, return inverse transformation of this coords.
         Parameters exponent (numbers. Number) - exponent value. If exponent equals -1, return in-
             verse transformation of this coords. In current, support only -1 case.
         Returns out - output.
         Return type skrobot.coordinates.Coordinates
Attributes
descendants
dimension
     Return dimension of this coordinate
         Returns len(self.translation) – dimension of this coordinate
         Return type int
dual_quaternion
     Property of DualQuaternion
     Return DualQuaternion representation of this coordinate.
         Returns DualQuaternion - DualQuaternion representation of this coordinate
         Return type skrobot.coordinates.dual_quaternion.DualQuaternion
interlocking_joint_pairs
     Interlocking joint pairs.
     pairs are [(joint0, joint1), ...] If users want to use interlocking joints, please overwrite this method.
joint_max_angles
joint_min_angles
larm
lleg
name
     Return this coordinate's name
```

```
Returns self._name – name of this coordinate
Return type str

parent
quaternion
Property of quaternion
```

Returns $\mathbf{q} - [w, x, y, z]$ quaternion

Return type numpy.ndarray

Examples

```
>>> from numpy import pi
>>> from skrobot.coordinates import make_coords
>>> c = make_coords()
>>> c.quaternion
array([1., 0., 0., 0.])
>>> c.rotate(pi / 3, 'y').rotate(pi / 5, 'z')
>>> c.quaternion
array([0.8236391 , 0.1545085 , 0.47552826, 0.26761657])
```

rarm

rleg

rotation

Return rotation matrix of this coordinates.

Returns self._rotation – 3x3 rotation matrix

Return type numpy.ndarray

Examples

translation

Return translation of this coordinates.

Returns self._translation – vector shape of (3,). unit is [m]

Return type numpy.ndarray

```
>>> from skrobot.coordinates import Coordinates
>>> c = Coordinates()
>>> c.translation
array([0., 0., 0.])
>>> c.translate([0.1, 0.2, 0.3])
>>> c.translation
array([0.1, 0.2, 0.3])
```

x_axis

Return x axis vector of this coordinates.

Returns axis – x axis.

Return type numpy.ndarray

y_axis

Return y axis vector of this coordinates.

Returns axis – y axis.

Return type numpy.ndarray

z_axis

Return z axis vector of this coordinates.

Returns axis – z axis.

Return type numpy.ndarray

3.2.2 Robot Model classes

You can create use robot model classes. Here is a example of robot models.

Fetch

skrobot.models.fetch.Fetch

Fetch Robot Model.

skrobot.models.fetch.Fetch

```
class skrobot.models.fetch.Fetch(*args, **kwargs)
    Fetch Robot Model.
```

http://docs.fetchrobotics.com/robot_hardware.html

Methods

T()

Return 4x4 homogeneous transformation matrix.

Returns matrix – homogeneous transformation matrix shape of (4, 4)

Return type numpy.ndarray

Examples

```
>>> from numpy import pi
>>> from skrobot.coordinates import make_coords
>>> c = make_coords()
>>> c.T()
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]
>>> c.translate([0.1, 0.2, 0.3])
>>> c.rotate(pi / 2.0, 'y')
array([[ 2.22044605e-16, 0.00000000e+00,
                                           1.00000000e+00,
         1.00000000e-01],
       [ 0.00000000e+00, 1.00000000e+00,
                                           0.00000000e+00.
         2.00000000e-01],
       [-1.000000000e+00, 0.00000000e+00,
                                           2.22044605e-16,
         3.00000000e-01],
       [ 0.00000000e+00, 0.0000000e+00,
                                           0.00000000e+00,
         1.00000000e+00]])
```

angle_vector(av=None, return_av=None)

Returns angle vector

If av is given, it updates angles of all joint. If given av violate min/max range, the value is modified.

```
assoc(child, relative coords='world', force=False, **kwargs)
```

Associate child coords to this coordinate system.

If *relative_coords* is *None* or 'world', the translation and rotation of childcoords in the world coordinate system do not change. If *relative_coords* is specified, childcoords is assoced at translation and rotation of *relative_coords*. By default, if child is already assoced to some other coords, raise an exception. But if *force* is *True*, you can overwrite the existing assoc relation.

Parameters

- child (CascadedCoords) child coordinates.
- relative_coords (None or Coordinates or str) child coordinates' relative coordinates.
- **force** (*bool*) predicate for overwriting the existing assoc-relation

Returns child – assoced child.

Return type CascadedCoords

```
>>> from skrobot.coordinates import CascadedCoords
>>> parent_coords = CascadedCoords(pos=[1, 0, 0])
>>> child_coords = CascadedCoords(pos=[1, 1, 0])
>>> parent_coords.assoc(child_coords)
#<CascadedCoords 0x7f1d30e29510 1.000 1.000 0.000 / 0.0 -0.0 0.0>
>>> child_coords.worldpos()
array([1., 1., 0.])
>>> child_coords.translation
array([0., 1., 0.])
```

None and 'world' have the same meaning.

```
>>> parent_coords = CascadedCoords(pos=[1, 0, 0])
>>> child_coords = CascadedCoords(pos=[1, 1, 0])
>>> parent_coords.assoc(child_coords, relative_coords='world')
#<CascadedCoords 0x7f1d30e29510 1.000 1.000 0.000 / 0.0 -0.0 0.0>
>>> child_coords.worldpos()
array([1., 1., 0.])
```

If *relative_coords* is 'local', *child* is associated at world translation and world rotation of *child* from this coordinate system.

```
>>> parent_coords = CascadedCoords(pos=[1, 0, 0])
>>> child_coords = CascadedCoords(pos=[1, 1, 0])
>>> parent_coords.assoc(child_coords, relative_coords='local')
>>> child_coords.worldpos()
array([2., 1., 0.])
>>> child_coords.translation
array([1., 1., 0.])
```

axis(ax)

```
calc_inverse_kinematics_nspace_from_link_list(link_list, avoid_nspace_gain=0.01,
```

union_link_list=None, n_joint_dimension=None, null_space=None, additional_nspace_list=None, weight=None)

Calculate all weight from link list.

```
calc_jacobian_for_interlocking_joints(link_list, interlocking_joint_pairs=None)
```

calc_jacobian_from_link_list(move_target, link_list=None, transform_coords=None,

rotation_axis=None, translation_axis=None, col_offset=0, dim=None, jacobian=None, additional_jacobi_dimension=0, n_joint_dimension=None, *args, **kwargs)

```
calc_joint_angle_from_min_max_table(index, j, av)
```

```
calc_joint_angle_speed_gain(union_link_list, dav, periodic_time)
calc_nspace_from_joint_limit(avoid_nspace_gain, union_link_list, weight)
     Calculate null-space according to joint limit.
calc_target_axis_dimension(rotation_axis, translation_axis)
     rotation-axis, translation-axis -> both list and atom OK.
calc_target_joint_dimension(link_list)
calc_union_link_list(link_list)
calc_vel_for_interlocking_joints(link_list, interlocking_joint_pairs=None)
     Calculate 0 velocity for keeping interlocking joint.
     at the same joint angle.
calc_vel_from_pos(dif_pos, translation_axis, p_limit=100.0)
     Calculate velocity from difference position
         Parameters
             • dif_pos (np.ndarray) - [m] order
             • translation_axis (str) - see calc dif with axis
         Returns vel_p
         Return type np.ndarray
calc_vel_from_rot(dif rot, rotation axis, r limit=0.5)
calc_weight_from_joint_limit(avoid_weight_gain, link_list, union_link_list, weight,
                                   n_joint_dimension=None)
     Calculate weight according to joint limit.
changed()
     Return False
     This is used for CascadedCoords compatibility
         Returns False – always return False
         Return type bool
collision_avoidance_link_pair_from_link_list(link_list, obstacles=None)
compute_qp_common(target_coords, move_target, dt, link_list=None, gain=0.85, weight=1.0,
                     translation_axis=True, rotation_axis=True, dof_limit_gain=0.5)
compute_velocity(target coords, move target, dt, link list=None, gain=0.85, weight=1.0,
                    translation axis=True, rotation axis=True, dof limit gain=0.5, fast=True,
                    sym proj=False, solver='cvxopt', *args, **kwargs)
coords()
     Return a deep copy of the Coordinates.
     Return a deep copy of the Coordinates.
copy_coords()
     Return a deep copy of the Coordinates.
copy_worldcoords()
     Return a deep copy of the Coordinates.
```

difference_position(coords, translation axis=True)

Return differences in position of given coords.

Parameters

- coords (skrobot.coordinates.Coordinates) given coordinates
- translation_axis (str or bool or None (optional)) we can take 'x', 'y', 'z', 'xy', 'yz', 'zx', 'xx', 'yy', 'zz', True or False(None).

Returns dif_pos – difference position of self coordinates and coords considering translation_axis.

Return type numpy.ndarray

Examples

difference_rotation(coords, rotation axis=True)

Return differences in rotation of given coords.

Parameters

- coords (skrobot.coordinates.Coordinates) given coordinates
- rotation_axis(str or bool or None (optional)) we can take 'x', 'y', 'z', 'xx', 'yy', 'zz', 'xm', 'ym', 'zm', 'xy', 'yx', 'yz', 'zy', 'zx', 'xz', True or False(None).

Returns dif_rot – difference rotation of self coordinates and coords considering rotation_axis.

Return type numpy.ndarray

Examples

```
>>> from numpy import pi
>>> from skrobot.coordinates import Coordinates
>>> from skrobot.coordinates.math import rpy_matrix
>>> coord1 = Coordinates()
>>> coord2 = Coordinates(rot=rpy_matrix(pi / 2.0, pi / 3.0, pi / 5.0))
>>> coord1.difference_rotation(coord2)
array([-0.32855112, 1.17434985, 1.05738936])
>>> coord1.difference_rotation(coord2, rotation_axis=False)
```

(continues on next page)

(continued from previous page)

```
array([0, 0, 0])
>>> coord1.difference_rotation(coord2, rotation_axis='x')
array([0. , 1.36034952, 0.78539816])
>>> coord1.difference_rotation(coord2, rotation_axis='y')
array([0.35398131, 0. , 0.97442695])
>>> coord1.difference_rotation(coord2, rotation_axis='z')
array([-0.88435715, 0.74192175, 0. ])
```

Using mirror option ['xm', 'ym', 'zm'], you can allow differences of mirror direction.

disable_hook()

dissoc(child)

```
find_joint_angle_limit_weight_from_union_link_list(union_link_list)
```

find_link_path(src_link, target_link)

Find paths of src_link to target_link

Parameters

- **src_link** (*skrobot.model.link.Link*) source link.
- target_link (skrobot.model.link.Link) target link.

Returns ret – If the links are connected, return Link list. Otherwise, return an empty list.

Return type List[skrobot.model.link.Link]

```
find_link_route(to, frm=None)
```

```
fix_leg_to_coords(fix_coords, leg='both', mid=0.5)
```

Fix robot's legs to a coords

In the Following codes, leged robot is assumed.

Parameters

- **fix_coords** (Coordinates) target coordinate
- leg (string) ['both', 'rleg', 'rleg', 'left', 'right']
- mid (float) ratio of legs coord.

get_transform()

Return Transform object

Returns transform – corrensponding Transform to this coordinates

Return type skrobot.coordinates.base.Transform

ik_convergence_check(dif_pos, dif_rot, rotation_axis, translation_axis, thre, rthre, centroid_thre=None, target_centroid_pos=None, centroid_offset_func=None, cog_translation_axis=None, update_mass_properties=True)

check ik convergence.

Parameters

- dif_pos (list of np.ndarray) -
- dif_rot (list of np.ndarray) -
- translation_axis (list of axis) -
- rotation_axis (list of axis) see _wrap_axis

init_pose()

inverse_kinematics(target_coords, move_target=None, link_list=None, **kwargs)

Solve inverse kinematics.

solve inverse kinematics, move move-target to target-coords look-at- target suppots t, nil, float-vector, coords, list of float-vector, list of coords link-list is set by default based on move-target -> root link link-list.

inverse_kinematics_loop(dif_pos, dif_rot, move_target, link_list=None, target_coords=None, **kwargs)
move move_target using dif_pos and dif_rot.

Parameters TODO -

Return type TODO

Solve look at inverse kinematics

Parameters look_at(list or np.ndarray or Coordinates) -

inverse_kinematics_optimization(target_coords, move_target=None, link_list=None,

regularization_parameter=None, init_angle_vector=None, translation_axis=True, rotation_axis=True, stop=100, dt=0.005, inverse_kinematics_hook=[], thre=0.001, rthre=0.017453292519943295, *args, **kwargs)

inverse_rotate_vector(v)

inverse_transform_vector(v)

Transform vector in world coordinates to local coordinates

Parameters vec (numpy.ndarray) – 3d vector. We can take batch of vector like (batch_size, 3)

Returns transformed_point – transformed point

Return type numpy.ndarray

inverse_transformation(dest=None)

Return a invese transformation of this coordinate system.

Create a new coordinate with inverse transformation of this coordinate system.

$$\left(\begin{array}{cc} R^{-1} & -R^{-1}p \\ 0 & 1 \end{array}\right)$$

Parameters dest (*None or* skrobot.coordinates.Coordinates) – If dest is given, the result of transformation is in-placed to dest.

```
Returns dest – result of inverse transformation.
```

Return type skrobot.coordinates.Coordinates

```
link_lists(to, frm=None)
```

Find link list from to link to frm link.

```
load_urdf(urdf)
```

```
load_urdf_file(file_obj)
```

look_at_hand(coords)

move_coords(target_coords, local_coords)

Transform this coordinate so that local_coords to target_coords.

Parameters

- target_coords (skrobot.coordinates.Coordinates) target coords.
- local_coords (skrobot.coordinates.Coordinates) local coords to be aligned.

Returns self.worldcoords() - world coordinates.

Return type *skrobot.coordinates.Coordinates*

```
move_end_pos(pos, wrt='local', *args, **kwargs)
```

move_end_rot(angle, axis, wrt='local', *args, **kwargs)

move_joints(union_vel, union_link_list=None, periodic_time=0.05, joint_args=None, move_joints_hook=None, *args, **kwargs)

obstacles=None, *args, **kwargs)

move_joints_avoidance(union_vel, union_link_list=None, link_list=None, n_joint_dimension=None, weight=None, null_space=None, avoid_nspace_gain=0.01, avoid_weight_gain=1.0, avoid_collision_distance=200, avoid_collision_null_gain=1.0, avoid_collision_joint_gain=1.0, collision_avoidance_link_pair=None, cog_gain=0.0, target_centroid_pos=None, centroid_offset_func=None, cog_translation_axis='z', cog_null_space=False, additional_weight_list=None, additional_nspace_list=None, jacobi=None,

newcoords(*c*, *pos=None*, *check_validity=True*)

Update this coordinates.

This function records that this CascadedCoords has changed and recursively records the change to descendants of this CascadedCoords.

Parameters

- **c** (skrobot.coordinates.Coordinates *or numpy.ndarray*) If pos is *None*, *c* means new Coordinates. If pos is given, *c* means rotation matrix.
- pos (numpy.ndarray or None) new translation.
- **check_validity** (*bool*) If this value is *True*, check whether an input rotation and an input translation are valid.

orient_with_matrix(rotation_matrix, wrt='world')

Force update this coordinate system's rotation.

Parameters

- **rotation_matrix** (*numpy.ndarray*) 3x3 rotation matrix.
- $\bullet \ \, \mathbf{wrt} \, (str \ or \ skrobot.coordinates. Coordinates) reference \, coordinates. \\$

```
parent_orientation(v, wrt)
parentcoords()
reset_joint_angle_limit_weight(union_link_list)
reset_manip_pose()
reset_pose()
rotate(theta, axis, wrt='local')
     Rotate this coordinate.
     Rotate this coordinate relative to axis by theta radians with respect to wrt.
         Parameters
             • theta (float) - radian
             • axis (str or numpy.ndarray) - 'x', 'y', 'z' or vector
             • wrt(str or Coordinates)-
         Return type self
rotate_vector(v)
     Rotate 3-dimensional vector using rotation of this coordinate
         Parameters v (numpy.ndarray) – vector shape of (3,)
         Returns np.matmul(self.rotation, v) – rotated vector
         Return type numpy.ndarray
     Examples
     >>> from skrobot.coordinates import Coordinates
     >>> from numpy import pi
     >>> c = Coordinates().rotate(pi, 'z')
     >>> c.rotate_vector([1, 2, 3])
     array([-1., -2., 3.])
rotate_with_matrix(matrix, wrt)
     Rotate this coordinate by given rotation matrix.
     This is a subroutine of self.rotate function.
         Parameters
             • mat (numpy.ndarray) – rotation matrix shape of (3, 3)
             • wrt (str or skrobot.coordinates.Coordinates) - with respect to.
         Returns self
         Return type skrobot.coordinates.Coordinates
rpy_angle()
     Return a pair of rpy angles of this coordinates.
         Returns rpy_angle(self.rotation)
                                                    pair
                                                           of
                                                                rpy
                                                                      angles.
                                                                                      See
                                                                                            also
             skrobot.coordinates.math.rpy_angle
         Return type tuple(numpy.ndarray, numpy.ndarray)
```

```
>>> import numpy as np
>>> from skrobot.coordinates import Coordinates
>>> c = Coordinates().rotate(np.pi / 2.0, 'x').rotate(np.pi / 3.0, 'z')
>>> r.rpy_angle()
(array([ 3.84592537e-16, -1.04719755e+00, 1.57079633e+00]),
array([ 3.14159265, -2.0943951 , -1.57079633]))
```

self_collision_check()

Return collision link pair

Returns

- is_collision (bool) True if a collision occurred between any pair of objects and False otherwise
- names (*set of 2-tuple*) The set of pairwise collisions. Each tuple contains two names in alphabetical order indicating that the two corresponding objects are in collision.

transform(c, wrt='local', out=None)

Transform this coordinates

Parameters

- c (skrobot.coordinates.Coordinates) coordinates
- wrt (str or skrobot.coordinates.Coordinates) transform this coordinates with respect to wrt. If wrt is 'local' or self, multiply c from the right. If wrt is 'parent' or self.parent, transform c with respect to parentcoords. (multiply c from the left.) If wrt is Coordinates, transform c with respect to c.
- **out** (*None or* skrobot.coordinates.Coordinates) If the *out* is specified, set new coordinates to *out*. Note that if the *out* is given, these coordinates don't change.

Returns self – return self

Return type skrobot.coordinates.CascadedCoords

transform_vector(v)

"Return vector represented at world frame.

Vector v given in the local coords is converted to world representation.

Parameters v (numpy.ndarray) – 3d vector. We can take batch of vector like (batch_size, 3)

Returns transformed_point - transformed point

Return type numpy.ndarray

transformation(c2, wrt='local')

translate(vec, wrt='local')

Translate this coordinates.

Note that this function changes this coordinates self. So if you don't want to change this class, use copy_worldcoords()

Parameters

- **vec** (list or numpy.ndarray) shape of (3,) translation vector. unit is [m] order.
- wrt (str or Coordinates (optional)) translate with respect to wrt.

```
>>> import numpy as np
    >>> from skrobot.coordinates import Coordinates
    >>> c = Coordinates()
    >>> c.translation
    array([0., 0., 0.], dtype=float32)
    >>> c.translate([0.1, 0.2, 0.3])
    >>> c.translation
    array([0.1, 0.2, 0.3], dtype=float32)
    >>> c = Coordinates()
    >>> c.copy_worldcoords().translate([0.1, 0.2, 0.3])
    >>> c.translation
    array([0., 0., 0.], dtype=float32)
    >>> c = Coordinates().rotate(np.pi / 2.0, 'y')
    >>> c.translate([0.1, 0.2, 0.3])
    >>> c.translation
    array([ 0.3, 0.2, -0.1])
    >>> c = Coordinates().rotate(np.pi / 2.0, 'y')
    >>> c.translate([0.1, 0.2, 0.3], 'world')
    >>> c.translation
    array([0.1, 0.2, 0.3])
update(force=False)
worldcoords()
    Calculate rotation and position in the world.
worldpos()
    Return translation of this coordinate
    See also skrobot.coordinates.Coordinates.translation
        Returns self.translation – translation of this coordinate
        Return type numpy.ndarray
worldrot()
    Return rotation of this coordinate
    See also skrobot.coordinates.Coordinates.rotation
        Returns self.rotation – rotation matrix of this coordinate
        Return type numpy.ndarray
__eq__(value,/)
    Return self==value.
__ne__(value,/)
    Return self!=value.
__lt__(value,/)
    Return self<value.
__le__(value,/)
```

Return self<=value.

```
__gt__(value,/)
     Return self>value.
__ge__(value,/)
     Return self>=value.
__mul__(other c)
     Return Transformed Coordinates.
     Note that this function creates new Coordinates and does not change translation and rotation, unlike trans-
     form function.
         Parameters other_c (skrobot.coordinates.Coordinates) - input coordinates.
         Returns out – transformed coordinates multiplied other_c from the right. T = T_{self}
             T_{\text{other}_c}.
         Return type skrobot.coordinates.Coordinates
__pow__(exponent)
     Return exponential homogeneous matrix.
     If exponent equals -1, return inverse transformation of this coords.
         Parameters exponent (numbers. Number) - exponent value. If exponent equals -1, return in-
             verse transformation of this coords. In current, support only -1 case.
         Returns \ out - {\tt output}.
         Return type skrobot.coordinates.Coordinates
Attributes
default_urdf_path
descendants
dimension
     Return dimension of this coordinate
         Returns len(self.translation) – dimension of this coordinate
         Return type int
dual_quaternion
     Property of DualQuaternion
     Return DualQuaternion representation of this coordinate.
         Returns DualQuaternion - DualQuaternion representation of this coordinate
         Return type skrobot.coordinates.dual_quaternion.DualQuaternion
interlocking_joint_pairs
     Interlocking joint pairs.
     pairs are [(joint0, joint1), ...] If users want to use interlocking joints, please overwrite this method.
joint_max_angles
joint_min_angles
larm
lleg
```

name

Return this coordinate's name

Returns self._name – name of this coordinate

Return type str

parent

quaternion

Property of quaternion

Returns $\mathbf{q} - [\mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z}]$ quaternion

Return type numpy.ndarray

Examples

```
>>> from numpy import pi
>>> from skrobot.coordinates import make_coords
>>> c = make_coords()
>>> c.quaternion
array([1., 0., 0., 0.])
>>> c.rotate(pi / 3, 'y').rotate(pi / 5, 'z')
>>> c.quaternion
array([0.8236391 , 0.1545085 , 0.47552826, 0.26761657])
```

rarm

rleg

rotation

Return rotation matrix of this coordinates.

Returns self._rotation – 3x3 rotation matrix

Return type numpy.ndarray

Examples

translation

Return translation of this coordinates.

Returns self._translation – vector shape of (3,). unit is [m]

Return type numpy.ndarray

Examples

```
>>> from skrobot.coordinates import Coordinates
>>> c = Coordinates()
>>> c.translation
array([0., 0., 0.])
>>> c.translate([0.1, 0.2, 0.3])
>>> c.translation
array([0.1, 0.2, 0.3])
```

x_axis

Return x axis vector of this coordinates.

Returns axis – x axis.

Return type numpy.ndarray

y_axis

Return y axis vector of this coordinates.

Returns axis – y axis.

Return type numpy.ndarray

z_axis

Return z axis vector of this coordinates.

Returns axis – z axis.

Return type numpy.ndarray

Kuka

skrobot.models.kuka.Kuka

Kuka Robot Model.

skrobot.models.kuka.Kuka

```
class skrobot.models.kuka.Kuka(*args, **kwargs)
    Kuka Robot Model.
```

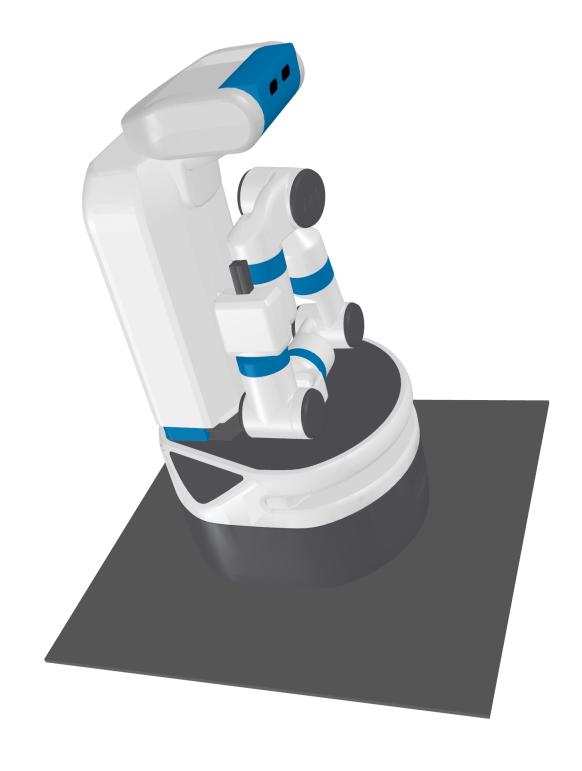
Methods

T()

Return 4x4 homogeneous transformation matrix.

Returns matrix – homogeneous transformation matrix shape of (4, 4)

Return type numpy.ndarray



```
>>> from numpy import pi
>>> from skrobot.coordinates import make_coords
>>> c = make_coords()
>>> c.T()
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]
>>> c.translate([0.1, 0.2, 0.3])
>>> c.rotate(pi / 2.0, 'y')
array([[ 2.22044605e-16, 0.00000000e+00,
                                            1.00000000e+00,
         1.00000000e-01],
       [0.000000000e+00, 1.00000000e+00,
                                            0.00000000e+00,
         2.00000000e-01],
       [-1.000000000e+00, 0.00000000e+00,
                                            2.22044605e-16,
         3.00000000e-01],
       [ 0.00000000e+00, 0.00000000e+00,
                                            0.00000000e+00,
         1.00000000e+00]])
```

angle_vector(av=None, return_av=None)

Returns angle vector

If av is given, it updates angles of all joint. If given av violate min/max range, the value is modified.

```
assoc(child, relative_coords='world', force=False, **kwargs)
```

Associate child coords to this coordinate system.

If *relative_coords* is *None* or 'world', the translation and rotation of childcoords in the world coordinate system do not change. If *relative_coords* is specified, childcoords is assoced at translation and rotation of *relative_coords*. By default, if child is already assoced to some other coords, raise an exception. But if *force* is *True*, you can overwrite the existing assoc relation.

Parameters

- child (CascadedCoords) child coordinates.
- relative_coords (None or Coordinates or str) child coordinates' relative coordinates.
- **force** (*bool*) predicate for overwriting the existing assoc-relation

Returns child - assoced child.

Return type CascadedCoords

Examples

```
>>> from skrobot.coordinates import CascadedCoords
>>> parent_coords = CascadedCoords(pos=[1, 0, 0])
>>> child_coords = CascadedCoords(pos=[1, 1, 0])
>>> parent_coords.assoc(child_coords)
#<CascadedCoords 0x7f1d30e29510 1.000 1.000 0.000 / 0.0 -0.0 0.0>
>>> child_coords.worldpos()
array([1., 1., 0.])
```

(continues on next page)

(continued from previous page)

```
>>> child_coords.translation
array([0., 1., 0.])
```

None and 'world' have the same meaning.

```
>>> parent_coords = CascadedCoords(pos=[1, 0, 0])
>>> child_coords = CascadedCoords(pos=[1, 1, 0])
>>> parent_coords.assoc(child_coords, relative_coords='world')
#<CascadedCoords 0x7f1d30e29510 1.000 1.000 0.000 / 0.0 -0.0 0.0>
>>> child_coords.worldpos()
array([1., 1., 0.])
```

If *relative_coords* is 'local', *child* is associated at world translation and world rotation of *child* from this coordinate system.

```
>>> parent_coords = CascadedCoords(pos=[1, 0, 0])
>>> child_coords = CascadedCoords(pos=[1, 1, 0])
>>> parent_coords.assoc(child_coords, relative_coords='local')
>>> child_coords.worldpos()
array([2., 1., 0.])
>>> child_coords.translation
array([1., 1., 0.])
```

axis(ax)

calc_inverse_kinematics_nspace_from_link_list(link_list, avoid_nspace_gain=0.01,

union_link_list=None, n_joint_dimension=None, null_space=None, additional_nspace_list=None, weight=None)

Calculate all weight from link list.

```
calc_jacobian_for_interlocking_joints(link_list, interlocking_joint_pairs=None)
```

n_joint_dimension=None, *args, **kwargs)

```
calc_joint_angle_from_min_max_table(index, j, av)
```

calc_joint_angle_speed_gain(union_link_list, dav, periodic_time)

calc_nspace_from_joint_limit(avoid_nspace_gain, union_link_list, weight)

Calculate null-space according to joint limit.

calc_target_axis_dimension(rotation_axis, translation_axis)

rotation-axis, translation-axis -> both list and atom OK.

calc_target_joint_dimension(link_list)

```
calc_union_link_list(link list)
calc_vel_for_interlocking_joints(link_list, interlocking_joint_pairs=None)
     Calculate 0 velocity for keeping interlocking joint.
     at the same joint angle.
calc_vel_from_pos(dif pos, translation axis, p limit=100.0)
     Calculate velocity from difference position
         Parameters
             • dif_pos (np.ndarray) – [m] order
             • translation_axis (str) - see calc_dif_with_axis
         Returns vel_p
         Return type np.ndarray
calc_vel_from_rot(dif_rot, rotation_axis, r_limit=0.5)
calc_weight_from_joint_limit(avoid_weight_gain, link_list, union_link_list, weight,
                                   n_joint_dimension=None)
     Calculate weight according to joint limit.
changed()
     Return False
     This is used for CascadedCoords compatibility
         Returns False – always return False
         Return type bool
close_hand(av=None)
collision_avoidance_link_pair_from_link_list(link_list, obstacles=None)
compute_qp_common(target_coords, move_target, dt, link_list=None, gain=0.85, weight=1.0,
                     translation_axis=True, rotation_axis=True, dof_limit_gain=0.5)
compute_velocity(target_coords, move_target, dt, link_list=None, gain=0.85, weight=1.0,
                    translation_axis=True, rotation_axis=True, dof_limit_gain=0.5, fast=True,
                    sym_proj=False, solver='cvxopt', *args, **kwargs)
coords()
     Return a deep copy of the Coordinates.
copy()
     Return a deep copy of the Coordinates.
copy_coords()
     Return a deep copy of the Coordinates.
copy_worldcoords()
     Return a deep copy of the Coordinates.
difference_position(coords, translation_axis=True)
     Return differences in position of given coords.
         Parameters
             • coords (skrobot.coordinates.Coordinates) – given coordinates
             • translation_axis (str or bool or None (optional)) - we can take 'x', 'y', 'z',
               'xy', 'yz', 'zx', 'xx', 'yy', 'zz', True or False(None).
```

Returns dif_pos – difference position of self coordinates and coords considering translation_axis.

Return type numpy.ndarray

Examples

```
>>> from skrobot.coordinates import Coordinates
>>> from skrobot.coordinates import transform_coords
>>> from numpy import pi
>>> c1 = Coordinates().translate([0.1, 0.2, 0.3]).rotate(
            pi / 3.0, 'x')
. . .
>>> c2 = Coordinates().translate([0.3, -0.3, 0.1]).rotate(
            pi / 2.0, 'y')
>>> c1.difference_position(c2)
array([ 0.2
            , -0.42320508, 0.3330127 ])
>>> c1 = Coordinates().translate([0.1, 0.2, 0.3]).rotate(0, 'x')
>>> c2 = Coordinates().translate([0.3, -0.3, 0.1]).rotate(
            pi / 3.0, 'x')
>>> c1.difference_position(c2)
array([ 0.2, -0.5, -0.2])
```

difference_rotation(coords, rotation_axis=True)

Return differences in rotation of given coords.

Parameters

- coords (skrobot.coordinates.Coordinates) given coordinates
- rotation_axis (str or bool or None (optional)) we can take 'x', 'y', 'z', 'xx', 'yy', 'zz', 'xm', 'ym', 'zm', 'xy', 'yx', 'yz', 'zy', 'zx', 'xz', True or False(None).

Returns dif_rot – difference rotation of self coordinates and coords considering rotation_axis.

Return type numpy.ndarray

Examples

```
>>> from numpy import pi
>>> from skrobot.coordinates import Coordinates
>>> from skrobot.coordinates.math import rpy_matrix
>>> coord1 = Coordinates()
>>> coord2 = Coordinates(rot=rpy_matrix(pi / 2.0, pi / 3.0, pi / 5.0))
>>> coord1.difference_rotation(coord2)
array([-0.32855112, 1.17434985, 1.05738936])
>>> coord1.difference_rotation(coord2, rotation_axis=False)
array([0, 0, 0])
>>> coord1.difference_rotation(coord2, rotation_axis='x')
                , 1.36034952, 0.78539816])
array([0.
>>> coord1.difference_rotation(coord2, rotation_axis='y')
array([0.35398131, 0.
                            , 0.97442695])
>>> coord1.difference_rotation(coord2, rotation_axis='z')
array([-0.88435715, 0.74192175, 0.
```

Using mirror option ['xm', 'ym', 'zm'], you can allow differences of mirror direction.

```
>>> coord1 = Coordinates()
     >>> coord2 = Coordinates().rotate(pi, 'x')
     >>> coord1.difference_rotation(coord2, 'xm')
     array([-2.99951957e-32, 0.00000000e+00, 0.00000000e+00])
     >>> coord1 = Coordinates()
     >>> coord2 = Coordinates().rotate(pi / 2.0, 'x')
     >>> coord1.difference_rotation(coord2, 'xm')
     array([-1.57079633, 0.
                                                        1)
disable_hook()
dissoc(child)
find_joint_angle_limit_weight_from_union_link_list(union_link_list)
find_link_path(src_link, target_link)
     Find paths of src_link to target_link
         Parameters
             • src_link (skrobot.model.link.Link) – source link.
             • target_link (skrobot.model.link.Link) - target link.
         Returns ret – If the links are connected, return Link list. Otherwise, return an empty list.
         Return type List[skrobot.model.link.Link]
find_link_route(to, frm=None)
fix_leg_to_coords(fix_coords, leg='both', mid=0.5)
     Fix robot's legs to a coords
     In the Following codes, leged robot is assumed.
         Parameters
             • fix_coords (Coordinates) – target coordinate
             • leg (string) - ['both', 'rleg', 'rleg', 'left', 'right']
             • mid (float) – ratio of legs coord.
get_transform()
     Return Transform object
         Returns transform – corrensponding Transform to this coordinates
         Return type skrobot.coordinates.base.Transform
ik_convergence_check(dif_pos, dif_rot, rotation_axis, translation_axis, thre, rthre, centroid_thre=None,
                        target_centroid_pos=None, centroid_offset_func=None,
                        cog_translation_axis=None, update_mass_properties=True)
     check ik convergence.
         Parameters
             • dif_pos (list of np.ndarray) -
             • dif_rot (list of np.ndarray) -
             • translation_axis(list of axis) -
             • rotation_axis (list of axis) - see _wrap_axis
init_pose()
```

inverse_kinematics(target_coords, move_target=None, link_list=None, **kwargs)
Solve inverse kinematics.

solve inverse kinematics, move move-target to target-coords look-at- target suppots t, nil, float-vector, coords, list of float-vector, list of coords link-list is set by default based on move-target -> root link link-list.

inverse_kinematics_loop(dif_pos, dif_rot, move_target, link_list=None, target_coords=None, **kwargs)
move move_target using dif_pos and dif_rot.

Parameters TODO -

Return type TODO

Solve look at inverse kinematics

Parameters look_at (list or np.ndarray or Coordinates) -

inverse_kinematics_optimization(target_coords, move_target=None, link_list=None,

regularization_parameter=None, init_angle_vector=None, translation_axis=True, rotation_axis=True, stop=100, dt=0.005, inverse_kinematics_hook=[], thre=0.001, rthre=0.017453292519943295, *args, **kwargs)

inverse_rotate_vector(v)

inverse_transform_vector(v)

Transform vector in world coordinates to local coordinates

Parameters vec (numpy.ndarray) – 3d vector. We can take batch of vector like (batch_size, 3)

Returns transformed_point - transformed point

Return type numpy.ndarray

inverse_transformation(dest=None)

Return a invese transformation of this coordinate system.

Create a new coordinate with inverse transformation of this coordinate system.

$$\left(\begin{array}{cc} R^{-1} & -R^{-1}p \\ 0 & 1 \end{array}\right)$$

Parameters dest (*None or* skrobot.coordinates.Coordinates) – If dest is given, the result of transformation is in-placed to dest.

Returns dest – result of inverse transformation.

Return type skrobot.coordinates.Coordinates

link_lists(to, frm=None)

Find link list from to link to frm link.

load_urdf(urdf)

load_urdf_file(file_obj)

look_at_hand(coords)

move_coords(target_coords, local_coords)

Transform this coordinate so that local_coords to target_coords.

Parameters

- target_coords (skrobot.coordinates.Coordinates) target coords.
- local_coords (skrobot.coordinates.Coordinates) local coords to be aligned.

Returns self.worldcoords() – world coordinates.

```
Return type skrobot.coordinates.Coordinates
```

```
move_joints_avoidance(union_vel, union_link_list=None, link_list=None, n_joint_dimension=None, weight=None, null_space=None, avoid_nspace_gain=0.01, avoid_weight_gain=1.0, avoid_collision_distance=200, avoid_collision_null_gain=1.0, avoid_collision_joint_gain=1.0, collision_avoidance_link_pair=None, cog_gain=0.0, target_centroid_pos=None, centroid_offset_func=None, cog_translation_axis='z', cog_null_space=False, additional_weight_list=None, additional_nspace_list=None, jacobi=None, obstacles=None, *args, **kwargs)
```

newcoords(c, pos=None, check_validity=True)

Update this coordinates.

This function records that this CascadedCoords has changed and recursively records the change to descendants of this CascadedCoords.

Parameters

- **c** (skrobot.coordinates.Coordinates *or numpy.ndarray*) If pos is *None*, *c* means new Coordinates. If pos is given, *c* means rotation matrix.
- pos (numpy.ndarray or None) new translation.
- **check_validity** (*bool*) If this value is *True*, check whether an input rotation and an input translation are valid.

```
open_hand(default_angle=0.17453292519943295, av=None)
```

```
orient_with_matrix(rotation matrix, wrt='world')
```

Force update this coordinate system's rotation.

Parameters

- rotation_matrix (numpy.ndarray) 3x3 rotation matrix.
- wrt(str or skrobot.coordinates.Coordinates) reference coordinates.

```
parent_orientation(v, wrt)
parentcoords()
reset_joint_angle_limit_weight(union_link_list)
reset_manip_pose()
reset_pose()
rotate(theta, axis, wrt='local')
    Rotate this coordinate.
```

Rotate this coordinate relative to axis by theta radians with respect to wrt.

Parameters

```
• theta (float) - radian
```

```
• axis (str or numpy.ndarray) - 'x', 'y', 'z' or vector
```

```
• wrt (str or Coordinates) -
```

Return type self

rotate_vector(v)

Rotate 3-dimensional vector using rotation of this coordinate

```
Parameters v (numpy.ndarray) – vector shape of (3,)
```

Returns np.matmul(self.rotation, v) – rotated vector

Return type numpy.ndarray

Examples

```
>>> from skrobot.coordinates import Coordinates
>>> from numpy import pi
>>> c = Coordinates().rotate(pi, 'z')
>>> c.rotate_vector([1, 2, 3])
array([-1., -2., 3.])
```

rotate_with_matrix(matrix, wrt)

Rotate this coordinate by given rotation matrix.

This is a subroutine of self.rotate function.

Parameters

- mat (numpy.ndarray) rotation matrix shape of (3, 3)
- wrt(str or skrobot.coordinates.Coordinates) with respect to.

Returns self

Return type *skrobot.coordinates.Coordinates*

rpy_angle()

Return a pair of rpy angles of this coordinates.

```
Returns rpy_angle(self.rotation) – a pair of rpy angles. See also skrobot.coordinates.math.rpy_angle
```

Return type tuple(numpy.ndarray, numpy.ndarray)

Examples

```
>>> import numpy as np
>>> from skrobot.coordinates import Coordinates
>>> c = Coordinates().rotate(np.pi / 2.0, 'x').rotate(np.pi / 3.0, 'z')
>>> r.rpy_angle()
(array([ 3.84592537e-16, -1.04719755e+00, 1.57079633e+00]),
array([ 3.14159265, -2.0943951 , -1.57079633]))
```

self_collision_check()

Return collision link pair

Returns

- is_collision (bool) True if a collision occurred between any pair of objects and False otherwise
- names (set of 2-tuple) The set of pairwise collisions. Each tuple contains two names in alphabetical order indicating that the two corresponding objects are in collision.

transform(c, wrt='local', out=None)

Transform this coordinates

Parameters

- c (skrobot.coordinates.Coordinates) coordinates
- wrt (str or skrobot.coordinates.Coordinates) transform this coordinates with respect to wrt. If wrt is 'local' or self, multiply c from the right. If wrt is 'parent' or self.parent, transform c with respect to parentcoords. (multiply c from the left.) If wrt is Coordinates, transform c with respect to c.
- **out** (*None or* skrobot.coordinates.Coordinates) If the *out* is specified, set new coordinates to *out*. Note that if the *out* is given, these coordinates don't change.

Returns self – return self

Return type skrobot.coordinates.CascadedCoords

transform_vector(v)

"Return vector represented at world frame.

Vector v given in the local coords is converted to world representation.

Parameters v (numpy.ndarray) – 3d vector. We can take batch of vector like (batch_size, 3)

Returns transformed_point - transformed point

Return type numpy.ndarray

transformation(c2, wrt='local')

translate(vec, wrt='local')

Translate this coordinates.

Note that this function changes this coordinates self. So if you don't want to change this class, use copy_worldcoords()

Parameters

- **vec** (1ist or numpy.ndarray) shape of (3,) translation vector. unit is [m] order.
- wrt (str or Coordinates (optional)) translate with respect to wrt.

Examples

```
>>> import numpy as np
    >>> from skrobot.coordinates import Coordinates
    >>> c = Coordinates()
    >>> c.translation
    array([0., 0., 0.], dtype=float32)
    >>> c.translate([0.1, 0.2, 0.3])
    >>> c.translation
    array([0.1, 0.2, 0.3], dtype=float32)
    >>> c = Coordinates()
    >>> c.copy_worldcoords().translate([0.1, 0.2, 0.3])
    >>> c.translation
    array([0., 0., 0.], dtype=float32)
    >>> c = Coordinates().rotate(np.pi / 2.0, 'y')
    >>> c.translate([0.1, 0.2, 0.3])
    >>> c.translation
    array([ 0.3, 0.2, -0.1])
    >>> c = Coordinates().rotate(np.pi / 2.0, 'y')
    >>> c.translate([0.1, 0.2, 0.3], 'world')
    >>> c.translation
    array([0.1, 0.2, 0.3])
update(force=False)
worldcoords()
    Calculate rotation and position in the world.
worldpos()
    Return translation of this coordinate
    See also skrobot.coordinates.Coordinates.translation
        Returns self.translation – translation of this coordinate
        Return type numpy.ndarray
worldrot()
    Return rotation of this coordinate
    See also skrobot.coordinates.Coordinates.rotation
        Returns self.rotation – rotation matrix of this coordinate
        Return type numpy.ndarray
__eq__(value,/)
    Return self==value.
__ne__(value,/)
    Return self!=value.
__lt__(value,/)
    Return self<value.
__le__(value,/)
```

Return self<=value.

```
__gt__(value,/)
     Return self>value.
__ge__(value,/)
     Return self>=value.
__mul__(other c)
     Return Transformed Coordinates.
     Note that this function creates new Coordinates and does not change translation and rotation, unlike trans-
     form function.
         Parameters other_c (skrobot.coordinates.Coordinates) - input coordinates.
         Returns out – transformed coordinates multiplied other_c from the right. T = T_{self}
             T_{\text{other}_c}.
         Return type skrobot.coordinates.Coordinates
__pow__(exponent)
     Return exponential homogeneous matrix.
     If exponent equals -1, return inverse transformation of this coords.
         Parameters exponent (numbers . Number) – exponent value. If exponent equals -1, return in-
             verse transformation of this coords. In current, support only -1 case.
         Returns \ out - {\tt output}.
         Return type skrobot.coordinates.Coordinates
Attributes
default_urdf_path
descendants
dimension
     Return dimension of this coordinate
         Returns len(self.translation) – dimension of this coordinate
         Return type int
dual_quaternion
     Property of DualQuaternion
     Return DualQuaternion representation of this coordinate.
         Returns DualQuaternion - DualQuaternion representation of this coordinate
         Return type skrobot.coordinates.dual_quaternion.DualQuaternion
interlocking_joint_pairs
     Interlocking joint pairs.
     pairs are [(joint0, joint1), ...] If users want to use interlocking joints, please overwrite this method.
joint_max_angles
joint_min_angles
larm
lleg
```

name

Return this coordinate's name

Returns self._name – name of this coordinate

Return type str

parent

quaternion

Property of quaternion

Returns $\mathbf{q} - [\mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z}]$ quaternion

Return type numpy.ndarray

Examples

```
>>> from numpy import pi
>>> from skrobot.coordinates import make_coords
>>> c = make_coords()
>>> c.quaternion
array([1., 0., 0., 0.])
>>> c.rotate(pi / 3, 'y').rotate(pi / 5, 'z')
>>> c.quaternion
array([0.8236391 , 0.1545085 , 0.47552826, 0.26761657])
```

rarm

rleg

rotation

Return rotation matrix of this coordinates.

Returns self._rotation – 3x3 rotation matrix

Return type numpy.ndarray

Examples

translation

Return translation of this coordinates.

Returns self._translation – vector shape of (3,). unit is [m]

Return type numpy.ndarray

Examples

```
>>> from skrobot.coordinates import Coordinates
>>> c = Coordinates()
>>> c.translation
array([0., 0., 0.])
>>> c.translate([0.1, 0.2, 0.3])
>>> c.translation
array([0.1, 0.2, 0.3])
```

x_axis

Return x axis vector of this coordinates.

Returns axis – x axis.

Return type numpy.ndarray

y_axis

Return y axis vector of this coordinates.

Returns axis – y axis.

Return type numpy.ndarray

z_axis

Return z axis vector of this coordinates.

Returns axis – z axis.

Return type numpy.ndarray

PR2

skrobot.models.pr2.PR2

PR2 Robot Model.

skrobot.models.pr2.PR2

```
class skrobot.models.pr2.PR2(*args, **kwargs)
    PR2 Robot Model.
```

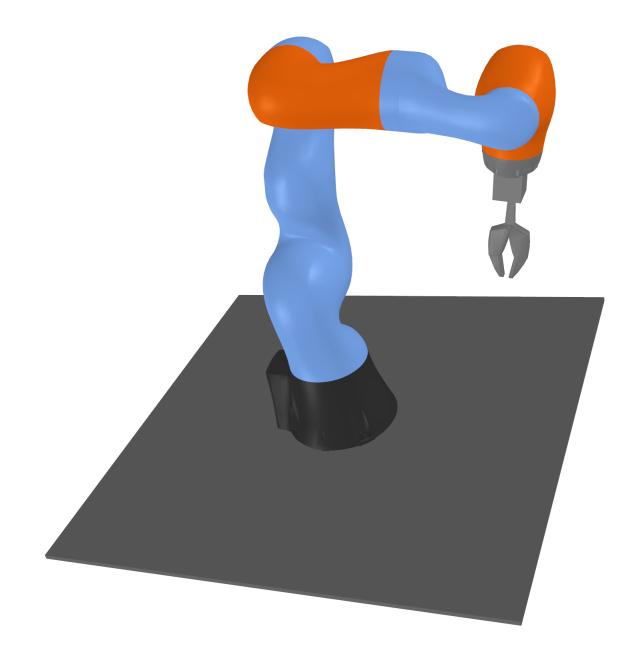
Methods

T()

Return 4x4 homogeneous transformation matrix.

Returns matrix – homogeneous transformation matrix shape of (4, 4)

Return type numpy.ndarray



Examples

```
>>> from numpy import pi
>>> from skrobot.coordinates import make_coords
>>> c = make_coords()
>>> c.T()
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]
>>> c.translate([0.1, 0.2, 0.3])
>>> c.rotate(pi / 2.0, 'y')
array([[ 2.22044605e-16, 0.00000000e+00,
                                            1.00000000e+00,
         1.00000000e-01],
       [0.000000000e+00, 1.00000000e+00,
                                            0.00000000e+00,
         2.00000000e-01],
       [-1.000000000e+00, 0.00000000e+00,
                                            2.22044605e-16,
         3.00000000e-01],
       [ 0.00000000e+00, 0.00000000e+00,
                                            0.00000000e+00,
         1.00000000e+00]])
```

angle_vector(av=None, return_av=None)

Returns angle vector

If av is given, it updates angles of all joint. If given av violate min/max range, the value is modified.

```
assoc(child, relative_coords='world', force=False, **kwargs)
```

Associate child coords to this coordinate system.

If *relative_coords* is *None* or 'world', the translation and rotation of childcoords in the world coordinate system do not change. If *relative_coords* is specified, childcoords is assoced at translation and rotation of *relative_coords*. By default, if child is already assoced to some other coords, raise an exception. But if *force* is *True*, you can overwrite the existing assoc relation.

Parameters

- child (CascadedCoords) child coordinates.
- relative_coords (None or Coordinates or str) child coordinates' relative coordinates.
- **force** (*bool*) predicate for overwriting the existing assoc-relation

Returns child - assoced child.

Return type CascadedCoords

Examples

```
>>> from skrobot.coordinates import CascadedCoords
>>> parent_coords = CascadedCoords(pos=[1, 0, 0])
>>> child_coords = CascadedCoords(pos=[1, 1, 0])
>>> parent_coords.assoc(child_coords)
#<CascadedCoords 0x7f1d30e29510 1.000 1.000 0.000 / 0.0 -0.0 0.0>
>>> child_coords.worldpos()
array([1., 1., 0.])
```

(continues on next page)

(continued from previous page)

```
>>> child_coords.translation
array([0., 1., 0.])
```

None and 'world' have the same meaning.

```
>>> parent_coords = CascadedCoords(pos=[1, 0, 0])
>>> child_coords = CascadedCoords(pos=[1, 1, 0])
>>> parent_coords.assoc(child_coords, relative_coords='world')
#<CascadedCoords 0x7f1d30e29510 1.000 1.000 0.000 / 0.0 -0.0 0.0>
>>> child_coords.worldpos()
array([1., 1., 0.])
```

If *relative_coords* is 'local', *child* is associated at world translation and world rotation of *child* from this coordinate system.

```
>>> parent_coords = CascadedCoords(pos=[1, 0, 0])
>>> child_coords = CascadedCoords(pos=[1, 1, 0])
>>> parent_coords.assoc(child_coords, relative_coords='local')
>>> child_coords.worldpos()
array([2., 1., 0.])
>>> child_coords.translation
array([1., 1., 0.])
```

axis(ax)

calc_inverse_kinematics_nspace_from_link_list(link_list, avoid_nspace_gain=0.01,

union_link_list=None, n_joint_dimension=None, null_space=None, additional_nspace_list=None, weight=None)

Calculate all weight from link list.

```
calc_jacobian_for_interlocking_joints(link_list, interlocking_joint_pairs=None)
```

n_joint_dimension=None, *args, **kwargs)

calc_joint_angle_from_min_max_table(index, j, av)

calc_joint_angle_speed_gain(union_link_list, dav, periodic_time)

calc_nspace_from_joint_limit(avoid_nspace_gain, union_link_list, weight)

Calculate null-space according to joint limit.

calc_target_axis_dimension(rotation_axis, translation_axis)

rotation-axis, translation-axis -> both list and atom OK.

calc_target_joint_dimension(link_list)

```
calc_union_link_list(link list)
calc_vel_for_interlocking_joints(link_list, interlocking_joint_pairs=None)
     Calculate 0 velocity for keeping interlocking joint.
     at the same joint angle.
calc_vel_from_pos(dif pos, translation axis, p limit=100.0)
     Calculate velocity from difference position
         Parameters
             • dif_pos (np.ndarray) – [m] order
             • translation_axis (str) - see calc_dif_with_axis
         Returns vel_p
         Return type np.ndarray
calc_vel_from_rot(dif_rot, rotation_axis, r_limit=0.5)
calc_weight_from_joint_limit(avoid_weight_gain, link_list, union_link_list, weight,
                                   n_joint_dimension=None)
     Calculate weight according to joint limit.
changed()
     Return False
     This is used for CascadedCoords compatibility
         Returns False – always return False
         Return type bool
collision_avoidance_link_pair_from_link_list(link_list, obstacles=None)
compute_qp_common(target_coords, move_target, dt, link_list=None, gain=0.85, weight=1.0,
                     translation_axis=True, rotation_axis=True, dof_limit_gain=0.5)
compute_velocity(target_coords, move_target, dt, link_list=None, gain=0.85, weight=1.0,
                    translation_axis=True, rotation_axis=True, dof_limit_gain=0.5, fast=True,
                    sym_proj=False, solver='cvxopt', *args, **kwargs)
coords()
     Return a deep copy of the Coordinates.
copy()
     Return a deep copy of the Coordinates.
copy_coords()
     Return a deep copy of the Coordinates.
copy_worldcoords()
     Return a deep copy of the Coordinates.
difference_position(coords, translation_axis=True)
     Return differences in position of given coords.
             • coords (skrobot.coordinates.Coordinates) – given coordinates
             • translation_axis(str or bool or None (optional)) - we can take 'x', 'y', 'z',
               'xy', 'yz', 'zx', 'xx', 'yy', 'zz', True or False(None).
```

Returns dif_pos – difference position of self coordinates and coords considering translation_axis.

Return type numpy.ndarray

Examples

```
>>> from skrobot.coordinates import Coordinates
>>> from skrobot.coordinates import transform_coords
>>> from numpy import pi
>>> c1 = Coordinates().translate([0.1, 0.2, 0.3]).rotate(
            pi / 3.0, 'x')
. . .
>>> c2 = Coordinates().translate([0.3, -0.3, 0.1]).rotate(
            pi / 2.0, 'y')
>>> c1.difference_position(c2)
array([ 0.2
            , -0.42320508, 0.3330127 ])
>>> c1 = Coordinates().translate([0.1, 0.2, 0.3]).rotate(0, 'x')
>>> c2 = Coordinates().translate([0.3, -0.3, 0.1]).rotate(
            pi / 3.0, 'x')
>>> c1.difference_position(c2)
array([ 0.2, -0.5, -0.2])
```

difference_rotation(coords, rotation_axis=True)

Return differences in rotation of given coords.

Parameters

- coords (skrobot.coordinates.Coordinates) given coordinates
- rotation_axis (str or bool or None (optional)) we can take 'x', 'y', 'z', 'xx', 'yy', 'zz', 'xm', 'ym', 'zm', 'xy', 'yx', 'yz', 'zy', 'zx', 'xz', True or False(None).

Returns dif_rot – difference rotation of self coordinates and coords considering rotation_axis.

Return type numpy.ndarray

Examples

```
>>> from numpy import pi
>>> from skrobot.coordinates import Coordinates
>>> from skrobot.coordinates.math import rpy_matrix
>>> coord1 = Coordinates()
>>> coord2 = Coordinates(rot=rpy_matrix(pi / 2.0, pi / 3.0, pi / 5.0))
>>> coord1.difference_rotation(coord2)
array([-0.32855112, 1.17434985, 1.05738936])
>>> coord1.difference_rotation(coord2, rotation_axis=False)
array([0, 0, 0])
>>> coord1.difference_rotation(coord2, rotation_axis='x')
                , 1.36034952, 0.78539816])
array([0.
>>> coord1.difference_rotation(coord2, rotation_axis='y')
array([0.35398131, 0.
                            , 0.97442695])
>>> coord1.difference_rotation(coord2, rotation_axis='z')
array([-0.88435715, 0.74192175, 0.
```

Using mirror option ['xm', 'ym', 'zm'], you can allow differences of mirror direction.

```
>>> coord1 = Coordinates()
>>> coord2 = Coordinates().rotate(pi, 'x')
>>> coord1.difference_rotation(coord2, 'xm')
array([-2.99951957e-32, 0.000000000e+00, 0.00000000e+00])
>>> coord1 = Coordinates()
>>> coord2 = Coordinates().rotate(pi / 2.0, 'x')
>>> coord1.difference_rotation(coord2, 'xm')
array([-1.57079633, 0. , 0. ])
```

disable_hook()

dissoc(child)

find_joint_angle_limit_weight_from_union_link_list(union_link_list)

find_link_path(src_link, target_link)

Find paths of src_link to target_link

Parameters

- **src_link** (*skrobot.model.link.Link*) source link.
- $\bullet \ \ target_link \ (skrobot.model.link.Link) target \ link.$

Returns ret – If the links are connected, return Link list. Otherwise, return an empty list.

Return type List[skrobot.model.link.Link]

find_link_route(to, frm=None)

fix_leg_to_coords(fix_coords, leg='both', mid=0.5)

Fix robot's legs to a coords

In the Following codes, leged robot is assumed.

Parameters

- **fix_coords** (Coordinates) target coordinate
- leg (string) ['both', 'rleg', 'rleg', 'left', 'right']
- mid (float) ratio of legs coord.

get_transform()

Return Transform object

Returns transform – corrensponding Transform to this coordinates

Return type skrobot.coordinates.base.Transform

```
gripper_distance(dist=None, arm='arms')
```

Change gripper angle function

Parameters

- **dist** (*None or float*) gripper distance. If dist is None, return gripper distance. If flaot value is given, change joint angle.
- arm (str) Specify target arm. You can specify 'larm', 'rarm', 'arms'.

Returns dist – Result of gripper distance in meter.

Return type float

ik_convergence_check(dif_pos, dif_rot, rotation_axis, translation_axis, thre, rthre, centroid_thre=None, target_centroid_pos=None, centroid_offset_func=None, cog_translation_axis=None, update_mass_properties=True)

check ik convergence.

Parameters

- dif_pos (list of np.ndarray) -
- dif_rot (list of np.ndarray) -
- translation_axis (list of axis) -
- rotation_axis (list of axis) see _wrap_axis

init_pose()

inverse_kinematics(target_coords, move_target=None, link_list=None, **kwargs)

Solve inverse kinematics.

solve inverse kinematics, move move-target to target-coords look-at- target suppots t, nil, float-vector, coords, list of float-vector, list of coords link-list is set by default based on move-target -> root link link-list.

inverse_kinematics_loop(dif_pos, dif_rot, move_target, link_list=None, target_coords=None, **kwargs)
move move_target using dif_pos and dif_rot.

Parameters TODO -

Return type TODO

Solve look at inverse kinematics

Parameters look_at(list or np.ndarray or Coordinates) -

inverse_kinematics_optimization(target_coords, move_target=None, link_list=None,

regularization_parameter=None, init_angle_vector=None, translation_axis=True, rotation_axis=True, stop=100, dt=0.005, inverse_kinematics_hook=[], thre=0.001, rthre=0.017453292519943295, *args, **kwargs)

inverse_rotate_vector(v)

inverse_transform_vector(v)

Transform vector in world coordinates to local coordinates

Parameters vec (numpy.ndarray) – 3d vector. We can take batch of vector like (batch_size, 3)

Returns transformed_point – transformed point

Return type numpy.ndarray

inverse_transformation(dest=None)

Return a invese transformation of this coordinate system.

Create a new coordinate with inverse transformation of this coordinate system.

$$\left(\begin{array}{cc} R^{-1} & -R^{-1}p \\ 0 & 1 \end{array}\right)$$

Parameters dest (*None or* skrobot.coordinates.Coordinates) – If dest is given, the result of transformation is in-placed to dest.

```
Returns dest – result of inverse transformation.
```

Return type skrobot.coordinates.Coordinates

```
link_lists(to, frm=None)
```

Find link list from to link to frm link.

```
load_urdf(urdf)
```

```
load_urdf_file(file_obj)
```

look_at_hand(coords)

move_coords(target_coords, local_coords)

Transform this coordinate so that local_coords to target_coords.

Parameters

- target_coords (skrobot.coordinates.Coordinates) target coords.
- local_coords (skrobot.coordinates.Coordinates) local coords to be aligned.

Returns self.worldcoords() - world coordinates.

Return type *skrobot.coordinates.Coordinates*

```
move_end_pos(pos, wrt='local', *args, **kwargs)
```

move_end_rot(angle, axis, wrt='local', *args, **kwargs)

move_joints_avoidance(union_vel, union_link_list=None, link_list=None, n_joint_dimension=None, weight=None, null_space=None, avoid_nspace_gain=0.01, avoid_weight_gain=1.0, avoid_collision_distance=200, avoid_collision_null_gain=1.0, avoid_collision_joint_gain=1.0, collision_avoidance_link_pair=None, cog_gain=0.0, target_centroid_pos=None, centroid_offset_func=None, cog_translation_axis='z', cog_null_space=False, additional_weight_list=None, additional_nspace_list=None, jacobi=None, obstacles=None, *args, **kwargs)

newcoords(*c*, *pos=None*, *check_validity=True*)

Update this coordinates.

This function records that this CascadedCoords has changed and recursively records the change to descendants of this CascadedCoords.

Parameters

- **c** (skrobot.coordinates.Coordinates *or numpy.ndarray*) If pos is *None*, *c* means new Coordinates. If pos is given, *c* means rotation matrix.
- pos (numpy.ndarray or None) new translation.
- **check_validity** (*bool*) If this value is *True*, check whether an input rotation and an input translation are valid.

orient_with_matrix(rotation_matrix, wrt='world')

Force update this coordinate system's rotation.

Parameters

- **rotation_matrix** (*numpy.ndarray*) 3x3 rotation matrix.
- $\bullet \ \, \mathbf{wrt} \, (str \ or \ skrobot.coordinates. Coordinates) reference \, coordinates. \\$

```
parent_orientation(v, wrt)
parentcoords()
reset_joint_angle_limit_weight(union_link_list)
reset_manip_pose()
reset_pose()
rotate(theta, axis, wrt='local')
    Rotate this coordinate.
     Rotate this coordinate relative to axis by theta radians with respect to wrt.
         Parameters
             • theta (float) - radian
             • axis (str or numpy.ndarray) - 'x', 'y', 'z' or vector
             • wrt(str or Coordinates)-
         Return type self
rotate_vector(v)
     Rotate 3-dimensional vector using rotation of this coordinate
         Parameters v (numpy.ndarray) – vector shape of (3,)
         Returns np.matmul(self.rotation, v) – rotated vector
         Return type numpy.ndarray
     Examples
     >>> from skrobot.coordinates import Coordinates
     >>> from numpy import pi
     >>> c = Coordinates().rotate(pi, 'z')
     >>> c.rotate_vector([1, 2, 3])
     array([-1., -2., 3.])
rotate_with_matrix(matrix, wrt)
     Rotate this coordinate by given rotation matrix.
     This is a subroutine of self.rotate function.
         Parameters
             • mat (numpy.ndarray) – rotation matrix shape of (3, 3)
             • wrt (str or skrobot.coordinates.Coordinates) - with respect to.
         Returns self
         Return type skrobot.coordinates.Coordinates
rpy_angle()
     Return a pair of rpy angles of this coordinates.
         Returns rpy_angle(self.rotation)
                                                    pair
                                                           of
                                                                rpy
                                                                      angles.
                                                                                      See
                                                                                            also
             skrobot.coordinates.math.rpy_angle
```

Return type tuple(numpy.ndarray, numpy.ndarray)

Examples

```
>>> import numpy as np
>>> from skrobot.coordinates import Coordinates
>>> c = Coordinates().rotate(np.pi / 2.0, 'x').rotate(np.pi / 3.0, 'z')
>>> r.rpy_angle()
(array([ 3.84592537e-16, -1.04719755e+00, 1.57079633e+00]),
array([ 3.14159265, -2.0943951 , -1.57079633]))
```

self_collision_check()

Return collision link pair

Returns

- is_collision (bool) True if a collision occurred between any pair of objects and False otherwise
- names (*set of 2-tuple*) The set of pairwise collisions. Each tuple contains two names in alphabetical order indicating that the two corresponding objects are in collision.

transform(c, wrt='local', out=None)

Transform this coordinates

Parameters

- c (skrobot.coordinates.Coordinates) coordinates
- wrt (str or skrobot.coordinates.Coordinates) transform this coordinates with respect to wrt. If wrt is 'local' or self, multiply c from the right. If wrt is 'parent' or self.parent, transform c with respect to parentcoords. (multiply c from the left.) If wrt is Coordinates, transform c with respect to c.
- **out** (*None or* skrobot.coordinates.Coordinates) If the *out* is specified, set new coordinates to *out*. Note that if the *out* is given, these coordinates don't change.

Returns self – return self

Return type skrobot.coordinates.CascadedCoords

transform_vector(v)

"Return vector represented at world frame.

Vector v given in the local coords is converted to world representation.

Parameters v (numpy.ndarray) – 3d vector. We can take batch of vector like (batch_size, 3)

Returns transformed_point - transformed point

Return type numpy.ndarray

transformation(c2, wrt='local')

translate(vec, wrt='local')

Translate this coordinates.

Note that this function changes this coordinates self. So if you don't want to change this class, use copy_worldcoords()

Parameters

- **vec** (list or numpy.ndarray) shape of (3,) translation vector. unit is [m] order.
- wrt (str or Coordinates (optional)) translate with respect to wrt.

Examples

```
>>> import numpy as np
    >>> from skrobot.coordinates import Coordinates
    >>> c = Coordinates()
    >>> c.translation
    array([0., 0., 0.], dtype=float32)
    >>> c.translate([0.1, 0.2, 0.3])
    >>> c.translation
    array([0.1, 0.2, 0.3], dtype=float32)
    >>> c = Coordinates()
    >>> c.copy_worldcoords().translate([0.1, 0.2, 0.3])
    >>> c.translation
    array([0., 0., 0.], dtype=float32)
    >>> c = Coordinates().rotate(np.pi / 2.0, 'y')
    >>> c.translate([0.1, 0.2, 0.3])
    >>> c.translation
    array([ 0.3, 0.2, -0.1])
    >>> c = Coordinates().rotate(np.pi / 2.0, 'y')
    >>> c.translate([0.1, 0.2, 0.3], 'world')
    >>> c.translation
    array([0.1, 0.2, 0.3])
update(force=False)
worldcoords()
    Calculate rotation and position in the world.
worldpos()
    Return translation of this coordinate
    See also skrobot.coordinates.Coordinates.translation
        Returns self.translation – translation of this coordinate
        Return type numpy.ndarray
worldrot()
    Return rotation of this coordinate
    See also skrobot.coordinates.Coordinates.rotation
        Returns self.rotation – rotation matrix of this coordinate
        Return type numpy.ndarray
__eq__(value,/)
    Return self==value.
__ne__(value,/)
    Return self!=value.
__lt__(value,/)
    Return self<value.
__le__(value,/)
```

Return self<=value.

```
__qt__(value,/)
     Return self>value.
__ge__(value,/)
     Return self>=value.
__mul__(other c)
     Return Transformed Coordinates.
     Note that this function creates new Coordinates and does not change translation and rotation, unlike trans-
     form function.
         Parameters other_c (skrobot.coordinates.Coordinates) - input coordinates.
         Returns out – transformed coordinates multiplied other_c from the right. T = T_{self}
             T_{\text{other}_c}.
         Return type skrobot.coordinates.Coordinates
__pow__(exponent)
     Return exponential homogeneous matrix.
     If exponent equals -1, return inverse transformation of this coords.
         Parameters exponent (numbers. Number) - exponent value. If exponent equals -1, return in-
             verse transformation of this coords. In current, support only -1 case.
         Returns out - output.
         Return type skrobot.coordinates.Coordinates
Attributes
default_urdf_path
descendants
dimension
     Return dimension of this coordinate
         Returns len(self.translation) – dimension of this coordinate
         Return type int
dual_quaternion
     Property of DualQuaternion
     Return DualQuaternion representation of this coordinate.
         Returns DualQuaternion - DualQuaternion representation of this coordinate
         Return type skrobot.coordinates.dual_quaternion.DualQuaternion
head
interlocking_joint_pairs
     Interlocking joint pairs.
     pairs are [(joint0, joint1), ...] If users want to use interlocking joints, please overwrite this method.
joint_max_angles
joint_min_angles
larm
```

lleg

name

Return this coordinate's name

Returns self._name – name of this coordinate

Return type str

parent

quaternion

Property of quaternion

Returns $\mathbf{q} - [\mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z}]$ quaternion

Return type numpy.ndarray

Examples

```
>>> from numpy import pi
>>> from skrobot.coordinates import make_coords
>>> c = make_coords()
>>> c.quaternion
array([1., 0., 0., 0.])
>>> c.rotate(pi / 3, 'y').rotate(pi / 5, 'z')
>>> c.quaternion
array([0.8236391 , 0.1545085 , 0.47552826, 0.26761657])
```

rarm

rleg

rotation

Return rotation matrix of this coordinates.

Returns self._rotation – 3x3 rotation matrix

Return type numpy.ndarray

Examples

translation

Return translation of this coordinates.

```
Returns self._translation – vector shape of (3, ). unit is [m] Return type numpy.ndarray
```

Examples

```
>>> from skrobot.coordinates import Coordinates
>>> c = Coordinates()
>>> c.translation
array([0., 0., 0.])
>>> c.translate([0.1, 0.2, 0.3])
>>> c.translation
array([0.1, 0.2, 0.3])
```

x_axis

Return x axis vector of this coordinates.

Returns axis – x axis.

Return type numpy.ndarray

y_axis

Return y axis vector of this coordinates.

Returns axis – y axis.

Return type numpy.ndarray

z_axis

Return z axis vector of this coordinates.

Returns axis - z axis.

Return type numpy.ndarray

3.3 Functions

3.3.1 Utilities functions

skrobot.coordinates.mathwrap_axis	Convert axis to float vector.
skrobot.coordinates.math.	Checks that the given rotation matrix is valid.
_check_valid_rotation	
skrobot.coordinates.math.	Checks that the translation vector is valid.
_check_valid_translation	
skrobot.coordinates.math.triple_product	Returns Triple Product
skrobot.coordinates.math.inverse_rodrigues	Inverse Rodrigues formula Convert Rotation-Matirx to
	Axis-Angle.
skrobot.coordinates.math.rotation_angle	Inverse Rodrigues formula Convert Rotation-Matirx to
	Axis-Angle.
skrobot.coordinates.math.make_matrix	Wrapper of numpy array.
skrobot.coordinates.math.random_rotation	Generates a random 3x3 rotation matrix.
skrobot.coordinates.math.	Generates a random translation vector.
random_translation	

continues on next page

3.3. Functions 93

skrobot.coordinates.math.midpoint

Table	8 –	continued	from	previous	page

Return midpoint

Returns mid (or p) rotation matrix of given two matrix	
r1 and r2.	
Return transform m v	
Return the rotation matrix.	
Rotate vector.	
Return rotation matrix from yaw-pitch-roll	
Decomposing a rotation matrix to yaw-pitch-roll.	
Return normalized vector	
Returns matrix log of given rotation matrix, it returns	
[-pi, pi]	
Returns exponent of given omega.	
Returns outer product matrix of given v.	
Returns Rotation matrix from yaw-pitch-roll angles.	
Return rotation matrix orienting first_axis	
Rodrigues formula.	
Inverse Rodrigues formula Convert Rotation-Matirx to	
Axis-Angle.	
Return the distance of rotation matrixes.	
Converts the rotation of a matrix into axis-angle repre-	
sentation.	
Returns the smallest angle in radians between two vec-	
tors.	

skrobot.coordinates.math._wrap_axis

skrobot.coordinates.math._wrap_axis(axis)

Convert axis to float vector.

Parameters axis (list or numpy.ndarray or str or bool or None) — rotation axis indicated by number or string.

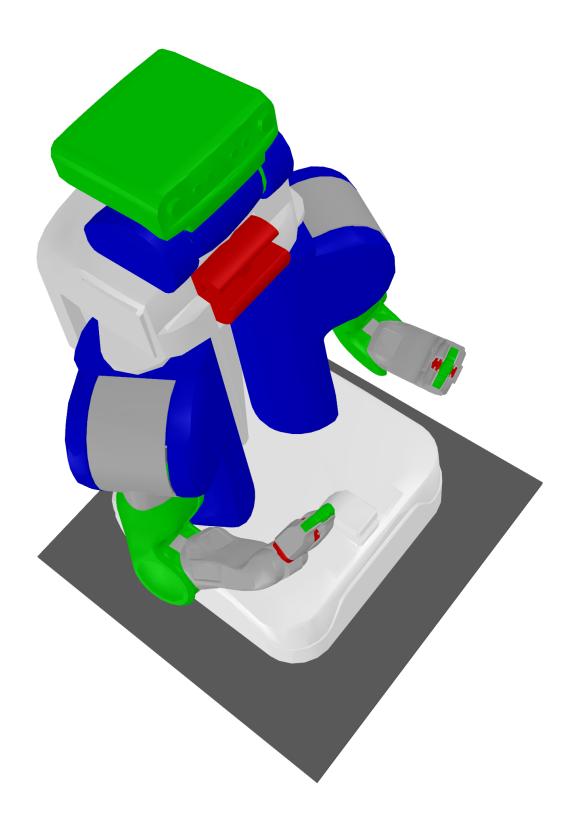
Returns axis – conveted axis

Return type numpy.ndarray

Examples

```
>>> from skrobot.coordinates.math import _wrap_axis
>>> _wrap_axis('x')
array([1, 0, 0])
>>> _wrap_axis('y')
array([0, 1, 0])
>>> _wrap_axis('z')
array([0, 0, 1])
>>> _wrap_axis('xy')
```

(continues on next page)



3.3. Functions 95

(continued from previous page)

```
array([1, 1, 0])
>>> _wrap_axis([1, 1, 1])
array([1, 1, 1])
>>> _wrap_axis(True)
array([0, 0, 0])
>>> _wrap_axis(False)
array([1, 1, 1])
```

skrobot.coordinates.math._check_valid_rotation

```
skrobot.coordinates.math._check_valid_rotation(rotation)
```

Checks that the given rotation matrix is valid.

skrobot.coordinates.math. check valid translation

```
skrobot.coordinates.math._check_valid_translation(translation)
```

Checks that the translation vector is valid.

skrobot.coordinates.math.triple_product

```
skrobot.coordinates.math.triple\_product(a, b, c)
```

Returns Triple Product

See https://en.wikipedia.org/wiki/Triple_product.

Geometrically, the scalar triple product

```
a \cdot (b \times c)
```

is the (signed) volume of the parallelepiped defined by the three vectors given.

Parameters

- a (numpy.ndarray) vector a
- **b** (numpy.ndarray) vector b
- c (numpy.ndarray) vector c

Returns triple product – calculated triple product

Return type float

Examples

```
>>> from skrobot.math import triple_product
>>> triple_product([1, 1, 1], [1, 1, 1], [1, 1, 1])
0
>>> triple_product([1, 0, 0], [0, 1, 0], [0, 0, 1])
1
```

skrobot.coordinates.math.inverse_rodrigues

```
skrobot.coordinates.math.inverse_rodrigues(mat)
```

Inverse Rodrigues formula Convert Rotation-Matirx to Axis-Angle.

Return theta and axis. If given unit matrix, return None.

```
Parameters mat (numpy.ndarray) – rotation matrix, shape (3, 3)
```

Returns theta, axis – rotation angle in radian and rotation axis

Return type tuple(float, numpy.ndarray)

Examples

```
>>> import numpy
>>> from skrobot.coordinates.math import rotation_angle
>>> rotation_angle(numpy.eye(3)) is None
True
>>> rotation_angle(numpy.array([[0, 0, 1], [0, 1, 0], [-1, 0, 0]]))
(1.5707963267948966, array([0., 1., 0.]))
```

skrobot.coordinates.math.rotation angle

```
skrobot.coordinates.math.rotation_angle(mat)
```

Inverse Rodrigues formula Convert Rotation-Matirx to Axis-Angle.

Return theta and axis. If given unit matrix, return None.

```
Parameters mat (numpy.ndarray) – rotation matrix, shape (3, 3)
```

Returns theta, axis – rotation angle in radian and rotation axis

Return type tuple(float, numpy.ndarray)

Examples

```
>>> import numpy
>>> from skrobot.coordinates.math import rotation_angle
>>> rotation_angle(numpy.eye(3)) is None
True
>>> rotation_angle(numpy.array([[0, 0, 1], [0, 1, 0], [-1, 0, 0]]))
(1.5707963267948966, array([[0., 1., 0.]]))
```

skrobot.coordinates.math.make_matrix

```
skrobot.coordinates.math.make_matrix(r, c) Wrapper of numpy array.
```

Parameters

- **r** (*int*) row of matrix
- c (int) column of matrix

3.3. Functions 97

```
Returns np.zeros((r, c), 'f') – matrix
Return type numpy.ndarray
```

skrobot.coordinates.math.random rotation

```
skrobot.coordinates.math.random_rotation()
Generates a random 3x3 rotation matrix.
```

Returns rot – randomly generated 3x3 rotation matrix

Return type numpy.ndarray

Examples

skrobot.coordinates.math.random_translation

```
{\tt skrobot.coordinates.math.random\_translation()}
```

Generates a random translation vector.

Returns translation – A 3-entry random translation vector.

Return type numpy.ndarray

Examples

```
>>> from skrobot.coordinates.math import random_translation
>>> random_translation()
array([0.03299473, 0.81481471, 0.57782565])
>>> random_translation()
array([0.10835455, 0.46549158, 0.73277675])
```

skrobot.coordinates.math.midpoint

```
skrobot.coordinates.math.midpoint(p, a, b)
Return midpoint
```

Parameters

- **p** (*float*) ratio of a:b
- a (numpy.ndarray) vector
- **b** (numpy.ndarray) vector

Returns midpoint – midpoint

Return type numpy.ndarray

Examples

```
>>> import numpy as np
>>> from skrobot.coordinates.math import midpoint
>>> midpoint(0.5, np.ones(3), np.zeros(3))
>>> array([0.5, 0.5, 0.5])
```

skrobot.coordinates.math.midrot

skrobot.coordinates.math.midrot(p, r1, r2)

Returns mid (or p) rotation matrix of given two matrix r1 and r2.

Parameters

- **p** (*float*) ratio of r1:r2
- **r1** (*numpy.ndarray*) 3x3 rotation matrix
- **r2** (*numpy.ndarray*) 3x3 rotation matrix

Returns $\mathbf{r} - 3x3$ rotation matrix

Return type numpy.ndarray

Examples

```
>>> import numpy as np
>>> from skrobot.coordinates.math import midrot
>>> midrot(0.5,
       np.eye(3),
       np.array([[0, 0, 1], [0, 1, 0], [-1, 0, 0]]))
array([[ 0.70710678,  0. ,  0.70710678],
                  , 1.
                              , 0.
       Γ0.
      [-0.70710678, 0.
                              , 0.70710678]])
>>> from skrobot.coordinates.math import rpy_angle
>>> np.rad2deg(rpy_angle(midrot(0.5,
              np.eye(3),
              np.array([[0, 0, 1], [0, 1, 0], [-1, 0, 0]])))[0])
array([ 0., 45., 0.])
```

3.3. Functions 99

skrobot.coordinates.math.transform

```
skrobot.coordinates.math.transform(m, v)
Return transform m v
```

Parameters

- **m** (numpy.ndarray) 3 x 3 rotation matrix.
- **v** (numpy.ndarray or list) input vector.

Returns np.matmul(m, v) – transformed vector.

Return type numpy.ndarray

skrobot.coordinates.math.rotation_matrix

```
skrobot.coordinates.math.rotation_matrix(theta, axis)
```

Return the rotation matrix.

Return the rotation matrix associated with counterclockwise rotation about the given axis by theta radians.

Parameters

- theta (float) radian
- axis (str or list or numpy.ndarray) rotation axis such that 'x', 'y', 'z' [0, 0, 1], [0, 1, 0], [1, 0, 0]

Returns rot – rotation matrix about the given axis by theta radians.

Return type numpy.ndarray

Examples

skrobot.coordinates.math.rotate vector

```
skrobot.coordinates.math.rotate_vector(vec, theta, axis)
```

Rotate vector.

Rotate vec with respect to axis.

Parameters

- vec (list or numpy.ndarray) target vector
- theta (float) rotation angle

```
• axis (list or numpy.ndarray or str) - axis of rotation.
```

Returns rotated_vec – rotated vector.

Return type numpy.ndarray

Examples

```
>>> from numpy import pi

>>> from skrobot.coordinates.math import rotate_vector

>>> rotate_vector([1, 0, 0], pi / 6.0, [1, 0, 0])

array([1., 0., 0.])

>>> rotate_vector([1, 0, 0], pi / 6.0, [0, 1, 0])

array([ 0.8660254, 0. , -0.5 ])

>>> rotate_vector([1, 0, 0], pi / 6.0, [0, 0, 1])

array([ 0.8660254, 0.5 , 0. ])
```

skrobot.coordinates.math.rotate_matrix

skrobot.coordinates.math.rotate_matrix(matrix, theta, axis, world=None)

skrobot.coordinates.math.rpy_matrix

```
skrobot.coordinates.math.rpy_matrix(az, ay, ax)
```

Return rotation matrix from yaw-pitch-roll

This function creates a new rotation matrix which has been rotated ax radian around x-axis in WORLD, ay radian around y-axis in WORLD, and az radian around z axis in WORLD, in this order. These angles can be extracted by the rpy function.

Parameters

- **az** (*float*) rotated around z-axis(yaw) in radian.
- ay (float) rotated around y-axis(pitch) in radian.
- ax (float) rotated around x-axis(roll) in radian.

Returns \mathbf{r} – rotation matrix

Return type numpy.ndarray

Examples

3.3. Functions 101

skrobot.coordinates.math.rpy_angle

```
skrobot.coordinates.math.rpy_angle(matrix)
Decomposing a rotation matrix to yaw-pitch-roll.

Parameters matrix(list or numpy.ndarray) - 3x3 rotation matrix

Returns rpy - pair of rpy in yaw-pitch-roll order.

Return type tuple(numpy.ndarray, numpy.ndarray)
```

Examples

```
>>> import numpy as np
>>> from skrobot.coordinates.math import rpy_matrix
>>> from skrobot.coordinates.math import rpy_angle
>>> yaw = np.pi / 2.0
>>> pitch = np.pi / 3.0
>>> roll = np.pi / 6.0
>>> rot = rpy_matrix(yaw, pitch, roll)
>>> rpy_angle(rot)
(array([1.57079633, 1.04719755, 0.52359878]),
    array([ 4.71238898, 2.0943951 , -2.61799388]))
```

skrobot.coordinates.math.normalize_vector

```
skrobot.coordinates.math.normalize_vector(v, ord=2)
Return normalized vector
```

Parameters

- v(list or numpy.ndarray) vector
- ord (int (optional)) ord of np.linalg.norm

Returns \mathbf{v} – normalized vector

Return type numpy.ndarray

Examples

```
>>> from skrobot.coordinates.math import normalize_vector
>>> normalize_vector([1, 1, 1])
array([0.57735027, 0.57735027, 0.57735027])
>>> normalize_vector([0, 0, 0])
array([0., 0., 0.])
```

skrobot.coordinates.math.matrix_log

```
skrobot.coordinates.math.matrix_log(m)
Returns matrix log of given rotation matrix, it returns [-pi, pi]

Parameters m(list or numpy.ndarray) - 3x3 rotation matrix
Returns matrixlog - vector of shape (3, )
Return type numpy.ndarray
```

Examples

```
>>> import numpy as np
>>> from skrobot.coordinates.math import matrix_log
>>> matrix_log(np.eye(3))
array([0., 0., 0.])
```

skrobot.coordinates.math.matrix_exponent

```
skrobot.coordinates.math.matrix_exponent(omega, p=1.0)
Returns exponent of given omega.
```

This function is similar to cv2.Rodrigues. Convert rvec (which is log quaternion) to rotation matrix.

```
Parameters omega (list or numpy.ndarray) – vector of shape (3,)

Returns rot – exponential matrix of given omega

Return type numpy.ndarray
```

Examples

3.3. Functions 103

skrobot.coordinates.math.outer_product_matrix

skrobot.coordinates.math.outer_product_matrix(v)

Returns outer product matrix of given v.

Returns following outer product matrix.

$$\left(\begin{array}{ccc}
0 & -v_2 & v_1 \\
v_2 & 0 & -v_0 \\
-v_1 & v_0 & 0
\end{array}\right)$$

Parameters v (numpy.ndarray or list) - [x, y, z]

Returns matrix – 3x3 rotation matrix.

Return type numpy.ndarray

Examples

skrobot.coordinates.math.rotation matrix from rpy

skrobot.coordinates.math.rotation_matrix_from_rpy(rpy)

Returns Rotation matrix from yaw-pitch-roll angles.

Parameters rpy (numpy.ndarray or list) - Vector of yaw-pitch-roll angles in radian.

Returns rot – 3x3 rotation matrix

Return type numpy.ndarray

Examples

skrobot.coordinates.math.rotation_matrix_from_axis

```
skrobot.coordinates.math.rotation_matrix_from_axis(first\_axis=(1, 0, 0), second\_axis=(0, 1, 0), axes='xy')
```

Return rotation matrix orienting first_axis

Parameters

- first_axis (list or tuple or numpy.ndarray) direction of first axis
- **second_axis** (*list or tuple or numpy.ndarray*) direction of second axis. This input axis is normalized using Gram-Schmidt.
- axes (str) valid inputs are 'xy', 'yx', 'xz', 'zx', 'yz', 'zy'. first index indicates first_axis's axis.

Returns rotation matrix – Rotation matrix

Return type numpy.ndarray

Examples

skrobot.coordinates.math.rodrigues

skrobot.coordinates.math.rodrigues(axis, theta=None)

Rodrigues formula.

See: Rodrigues' rotation formula - Wikipedia.

See: Axis-angle representation - Wikipedia.

Parameters

- axis (numpy.ndarray or list) [x, y, z] vector. You can give axis-angle representation to axis if theta is None.
- theta (float or None (optional)) radian. If None is given, calculate theta from axis.

Returns mat – 3x3 rotation matrix

Return type numpy.ndarray

skrobot.coordinates.math.rotation_distance

skrobot.coordinates.math.rotation_distance(mat1, mat2)

Return the distance of rotation matrixes.

Parameters

```
    mat1 (list or numpy.ndarray) -
    mat2 (list or numpy.ndarray) - 3x3 matrix
```

Returns diff_theta – distance of rotation matrixes in radian.

Return type float

Examples

skrobot.coordinates.math.axis_angle_from_matrix

```
skrobot.coordinates.math.axis_angle_from_matrix(rotation)

Converts the rotation of a matrix into axis-angle representation.
```

Parameters rotation (*numpy.ndarray*) – 3x3 rotation matrix

Returns axis_angle - axis-angle representation of vector

Return type numpy.ndarray

skrobot.coordinates.math.angle_between_vectors

skrobot.coordinates.math.angle_between_vectors(v1, v2, normalize=True, directed=True)
Returns the smallest angle in radians between two vectors.

Parameters

- **v1** (numpy.ndarray, list[float] or tuple(float)) input vector.
- **v2** (numpy.ndarray, list[float] or tuple(float)) input vector.
- **normalize** (*bool*) If normalize is True, normalize v1 and v2.
- **directed** (boo1) If directed is *False*, the input vectors are interpreted as undirected axes.

Returns theta – smallest angle between v1 and v2.

Return type float

3.3.2 Jacobian Functions

skrobot.coordinates.math.sr_inverse	Returns SR-inverse of given Jacobian.
skrobot.coordinates.math.sr_inverse_org	Return SR-inverse of given J
skrobot.coordinates.math.manipulability	Return manipulability of given matrix.

skrobot.coordinates.math.sr_inverse

```
skrobot.coordinates.math.sr_inverse(J, k=1.0, weight_vector=None)
Returns SR-inverse of given Jacobian.
```

Calculate Singularity-Robust Inverse See: Inverse Kinematic Solutions With Singularity Robustness for Robot Manipulator Control

Parameters

- **J** (numpy.ndarray) jacobian
- **k** (float) coefficients
- weight_vector (None or numpy.ndarray) weight vector

Returns ret – result of SR-inverse

Return type numpy.ndarray

skrobot.coordinates.math.sr_inverse_org

skrobot.coordinates.math.sr_inverse_org(J, k=1.0)

Return SR-inverse of given J

Definition of SR-inverse is following.

$$J^* = J^T (JJ^T + kI_m)^{-1}$$

Parameters

- **J** (numpy.ndarray) jacobian
- **k** (*float*) coefficients

Returns sr_inverse - calculated SR-inverse

Return type numpy.ndarray

skrobot.coordinates.math.manipulability

 ${\tt skrobot.coordinates.math.} \textbf{manipulability}(J)$

Return manipulability of given matrix.

Definition of manipulability is following.

$$w = \sqrt{\det J(\theta)J^T(\theta)}$$

Parameters J (numpy.ndarray) - jacobian

Returns w – manipulability

Return type float

3.3.3 Quaternion Functions

skrobot.coordinates.math.xyzw2wxyz	Convert quaternion [x, y, z, w] to [w, x, y, z] order.
skrobot.coordinates.math.wxyz2xyzw	Convert quaternion [w, x, y, z] to [x, y, z, w] order.
skrobot.coordinates.math.random_quaternion	Generate uniform random unit quaternion.
skrobot.coordinates.math.	Return multiplication of two quaternions.
quaternion_multiply	
skrobot.coordinates.math.	Return conjugate of quaternion.
quaternion_conjugate	
skrobot.coordinates.math.	Return inverse of quaternion.
quaternion_inverse	
skrobot.coordinates.math.quaternion_slerp	Return spherical linear interpolation between two
	quaternions.
skrobot.coordinates.math.	Return the distance of quaternion.
quaternion_distance	
skrobot.coordinates.math.	Return the absolute distance of quaternion.
quaternion_absolute_distance	
skrobot.coordinates.math.quaternion_norm	Return the norm of quaternion.
skrobot.coordinates.math.	Return the normalized quaternion.
quaternion_normalize	
skrobot.coordinates.math.matrix2quaternion	Returns quaternion of given rotation matrix.
skrobot.coordinates.math.quaternion2matrix	Returns matrix of given quaternion.
	continues on next page

continues on next page

Table 10 – continued from previous page

skrobot.coordinates.math.quaternion2rpy	Returns Roll-pitch-yaw angles of a given quaternion.
skrobot.coordinates.math.rpy2quaternion	Return Quaternion from yaw-pitch-roll angles.
skrobot.coordinates.math.rpy_from_quat	Returns Roll-pitch-yaw angles of a given quaternion.
skrobot.coordinates.math.	Returns quaternion of given rotation matrix.
quat_from_rotation_matrix	
skrobot.coordinates.math.quat_from_rpy	Return Quaternion from yaw-pitch-roll angles.
skrobot.coordinates.math.	Returns matrix of given quaternion.
rotation_matrix_from_quat	
skrobot.coordinates.math.	Return the quaternion from axis angle
quaternion_from_axis_angle	
skrobot.coordinates.math.	Converts a quaternion into the axis-angle representation.
<pre>axis_angle_from_quaternion</pre>	

skrobot.coordinates.math.xyzw2wxyz

```
skrobot.coordinates.math.xyzw2wxyz(quat)
Convert quaternion [x, y, z, w] to [w, x, y, z] order.

Parameters quat (list or numpy.ndarray) - quaternion [x, y, z, w]

Returns quaternion - quaternion [w, x, y, z]

Return type numpy.ndarray
```

Examples

```
>>> from skrobot.coordinates.math import xyzw2wxyz
>>> xyzw2wxyz([1, 2, 3, 4])
array([4, 1, 2, 3])
```

skrobot.coordinates.math.wxyz2xyzw

```
skrobot.coordinates.math.wxyz2xyzw(quat)
Convert quaternion [w, x, y, z] to [x, y, z, w] order.

Parameters quat (list or numpy.ndarray) - quaternion [w, x, y, z]

Returns quaternion - quaternion [x, y, z, w]

Return type numpy.ndarray
```

Examples

```
>>> from skrobot.coordinates.math import wxyz2xyzw
>>> wxyz2xyzw([1, 2, 3, 4])
array([2, 3, 4, 1])
```

skrobot.coordinates.math.random_quaternion

```
skrobot.coordinates.math.random_quaternion()
Generate uniform random unit quaternion.
```

Returns quaternion – generated random unit quaternion [w, x, y, z]

Return type numpy.ndarray

Examples

```
>>> from skrobot.coordinates.math import random_quaternion
>>> random_quaternion()
array([-0.02156994,  0.5404561 , -0.72781116, -0.42158374])
>>> random_quaternion()
array([-0.47302116,  0.020306 , -0.37539238,  0.79681818])
>>> from skrobot.coordinates.math import quaternion_norm
>>> q = random_quaternion()
>>> numpy.allclose(1.0, quaternion_norm(q))
True
>>> q.shape
(4,)
```

skrobot.coordinates.math.quaternion_multiply

skrobot.coordinates.math.quaternion_multiply(quaternion1, quaternion0) Return multiplication of two quaternions.

Parameters

- quaternion(list or numpy.ndarray) [w, x, y, z]
- quaternion1 (list or numpy.ndarray) [w, x, y, z]

Returns quaternion -[w, x, y, z]

Return type numpy.ndarray

Examples

```
>>> q = quaternion_multiply([4, 1, -2, 3], [8, -5, 6, 7])
>>> numpy.allclose(q, [28, -44, -14, 48])
True
```

skrobot.coordinates.math.quaternion_conjugate

Return type numpy.ndarray

```
skrobot.coordinates.math.quaternion_conjugate(quaternion)
    Return conjugate of quaternion.

Parameters quaternion(list or numpy.ndarray) - quaternion [w, x, y, z]

Returns conjugate of quaternion - [w, x, y, z]
```

Examples

```
>>> q0 = random_quaternion()
>>> q1 = quaternion_conjugate(q0)
>>> np.allclose(quaternion_multiply(q0, q1), [1.0, 0, 0, 0])
True
```

skrobot.coordinates.math.quaternion_inverse

```
skrobot.coordinates.math.quaternion_inverse(quaternion)
Return inverse of quaternion.
```

```
Parameters quaternion (list\ or\ numpy.ndarray) – [w, x, y, z]

Returns inverse of quaternion – [w, x, y, z]

Return type numpy.ndarray
```

Examples

```
>>> q0 = random_quaternion()
>>> q1 = quaternion_inverse(q0)
>>> np.allclose(quaternion_multiply(q0, q1), [1, 0, 0, 0])
True
```

skrobot.coordinates.math.quaternion_slerp

skrobot.coordinates.math.quaternion_slerp(q0, q1, fraction, spin=0, shortestpath=True)
Return spherical linear interpolation between two quaternions.

Parameters

```
    q0 (list or numpy.ndarray) – start quaternion
    q1 (list or numpy.ndarray) – end quaternion
    fraction (float) – ratio
    spin (int) – TODO
    shortestpath (bool) – TODO
```

Returns quaternion – spherical linear interpolated quaternion

Return type numpy.ndarray

skrobot.coordinates.math.quaternion distance

skrobot.coordinates.math.quaternion_distance(q1, q2, absolute=False)
Return the distance of quaternion.

Parameters

- q1(list or numpy.ndarray) -
- $\mathbf{q2}$ (list or numpy.ndarray) [w, x, y, z] order
- **absolute** (*bool*) if True, return distance accounting for the sign ambiguity.

Returns diff_theta – distance of q1 and q2 in radian.

Return type float

Examples

skrobot.coordinates.math.quaternion_absolute_distance

```
skrobot.coordinates.math.quaternion_absolute_distance(q1, q2) Return the absolute distance of quaternion.
```

Parameters

```
q1 (list or numpy.ndarray) -
q2 (list or numpy.ndarray) - [w, x, y, z] order
```

Returns diff_theta – absolute distance of q1 and q2 in radian.

Return type float

Examples

skrobot.coordinates.math.quaternion_norm

```
skrobot.coordinates.math.quaternion_norm(q)
Return the norm of quaternion.
```

```
\label{eq:parameters q (list or numpy.ndarray) - [w, x, y, z] order} \\ \mbox{\bf Returns norm}_{-} \mbox{\bf q} - \mbox{\bf quaternion norm of q} \\
```

Return type float

Examples

```
>>> from skrobot.coordinates.math import quaternion_norm
>>> q = [1, 1, 1, 1]
>>> quaternion_norm(q)
2.0
>>> q = [0, 0.7071067811865476, 0, 0.7071067811865476]
>>> quaternion_norm(q)
1.0
```

skrobot.coordinates.math.quaternion_normalize

```
{\tt skrobot.coordinates.math.} \textbf{quaternion\_normalize}(q)
```

Return the normalized quaternion.

```
Parameters q(list or numpy.ndarray) - [w, x, y, z] order
```

Returns normalized_q – normalized quaternion

Return type numpy.ndarray

Examples

```
>>> from skrobot.coordinates.math import quaternion_normalize
>>> from skrobot.coordinates.math import quaternion_norm
>>> q = quaternion_normalize([1, 1, 1, 1])
>>> quaternion_norm(q)
1.0
```

skrobot.coordinates.math.matrix2quaternion

```
skrobot.coordinates.math.matrix2quaternion(m)
```

Returns quaternion of given rotation matrix.

```
Parameters m (list or numpy.ndarray) – 3x3 rotation matrix
```

Returns quaternion – quaternion [w, x, y, z] order

Return type numpy.ndarray

Examples

```
>>> import numpy
>>> from skrobot.coordinates.math import matrix2quaternion
>>> matrix2quaternion(np.eye(3))
array([1., 0., 0., 0.])
```

skrobot.coordinates.math.quaternion2matrix

skrobot.coordinates.math.quaternion2matrix(q, normalize=False)Returns matrix of given quaternion.

Parameters

- quaternion (list or numpy.ndarray) quaternion [w, x, y, z] order
- **normalize** (*bool*) if normalize is True, input quaternion is normalized.

Returns rot – 3x3 rotation matrix

Return type numpy.ndarray

skrobot.coordinates.math.quaternion2rpy

```
{\tt skrobot.coordinates.math.quaternion2rpy}(q)
```

Returns Roll-pitch-yaw angles of a given quaternion.

Parameters q (numpy.ndarray or list) – Quaternion in [w x y z] format.

Returns rpy – Array of yaw-pitch-roll angles, in radian.

Return type numpy.ndarray

Examples

skrobot.coordinates.math.rpy2quaternion

```
skrobot.coordinates.math.rpy2quaternion(rpy)
```

Return Quaternion from yaw-pitch-roll angles.

Parameters rpy (numpy.ndarray or list) – Vector of yaw-pitch-roll angles in radian.

Returns quat – Quaternion in [w x y z] format.

Return type numpy.ndarray

Examples

```
>>> import numpy as np
>>> from skrobot.coordinates.math import rpy2quaternion
>>> rpy2quaternion([0, 0, 0])
array([1., 0., 0., 0.])
>>> yaw = np.pi / 3.0
>>> rpy2quaternion([yaw, 0, 0])
array([0.8660254, 0. , 0. , 0.5 ])
>>> rpy2quaternion([np.pi * 2 - yaw, 0, 0])
array([-0.8660254, -0. , 0. , 0.5 ])
```

skrobot.coordinates.math.rpy_from_quat

```
{\tt skrobot.coordinates.math.rpy\_from\_quat}(q)
```

Returns Roll-pitch-yaw angles of a given quaternion.

Parameters q (numpy.ndarray or list) - Quaternion in [w x y z] format.

Returns rpy – Array of yaw-pitch-roll angles, in radian.

Return type numpy.ndarray

Examples

skrobot.coordinates.math.quat_from_rotation_matrix

```
skrobot.coordinates.math.quat_from_rotation_matrix(m)
```

Returns quaternion of given rotation matrix.

Parameters m (list or numpy.ndarray) – 3x3 rotation matrix

Returns quaternion – quaternion [w, x, y, z] order

Return type numpy.ndarray

Examples

```
>>> import numpy
>>> from skrobot.coordinates.math import matrix2quaternion
>>> matrix2quaternion(np.eye(3))
array([1., 0., 0., 0.])
```

skrobot.coordinates.math.quat_from_rpy

```
skrobot.coordinates.math.quat_from_rpy(rpy)
```

Return Quaternion from yaw-pitch-roll angles.

Parameters rpy (numpy.ndarray or list) – Vector of yaw-pitch-roll angles in radian.

Returns quat – Quaternion in [w x y z] format.

Return type numpy.ndarray

```
>>> import numpy as np
>>> from skrobot.coordinates.math import rpy2quaternion
>>> rpy2quaternion([0, 0, 0])
array([1., 0., 0., 0.])
>>> yaw = np.pi / 3.0
>>> rpy2quaternion([yaw, 0, 0])
array([0.8660254, 0. , 0. , 0.5 ])
>>> rpy2quaternion([np.pi * 2 - yaw, 0, 0])
array([-0.8660254, -0. , 0. , 0.5 ])
```

skrobot.coordinates.math.rotation_matrix_from_quat

```
skrobot.coordinates.math.rotation_matrix_from_quat(q, normalize=False)
Returns matrix of given quaternion.
```

Parameters

- quaternion (list or numpy.ndarray) quaternion [w, x, y, z] order
- **normalize** (*bool*) if normalize is True, input quaternion is normalized.

Returns rot – 3x3 rotation matrix

Return type numpy.ndarray

Examples

skrobot.coordinates.math.quaternion from axis angle

```
skrobot.coordinates.math.quaternion_from_axis_angle(theta, axis)
```

Return the quaternion from axis angle

This function returns quaternion associated with counterclockwise rotation about the given axis by theta radians.

Parameters

- theta (float) radian
- axis (list or numpy.ndarray) length is 3. Automatically normalize in this function

Returns quaternion – [w, x, y, z] order

Return type numpy.ndarray

```
>>> import numpy
>>> from skrobot.coordinates.math import quaternion_from_axis_angle
>>> quaternion_from_axis_angle(0.1, [1, 0, 0])
array([0.99875026, 0.04997917, 0. , 0. ])
>>> quaternion_from_axis_angle(numpy.pi, [1, 0, 0])
array([6.123234e-17, 1.0000000e+00, 0.0000000e+00, 0.000000e+00])
>>> quaternion_from_axis_angle(0, [1, 0, 0])
array([1., 0., 0., 0.])
>>> quaternion_from_axis_angle(numpy.pi, [1, 0, 1])
array([6.12323400e-17, 7.07106781e-01, 0.000000000e+00, 7.07106781e-01])
```

skrobot.coordinates.math.axis_angle_from_quaternion

```
{\tt skrobot.coordinates.math.axis\_angle\_from\_quaternion}({\it quat})
```

Converts a quaternion into the axis-angle representation.

```
Parameters quat (numpy.ndarray) – quaternion [w, x, y, z]
```

Returns axis_angle – axis-angle representation of vector

Return type numpy.ndarray

Examples

```
>>> from skrobot.coordinates.math import axis_angle_from_quaternion
>>> axis_angle_from_quaternion([1, 0, 0, 0])
array([0, 0, 0])
>>> axis_angle_from_quaternion([0, 7.07106781e-01, 0, 7.07106781e-01])
array([2.22144147, 0. , 2.22144147])
```

3.3.4 Geometry functions

skrobot.coordinates.geo.rotate_points Rotate given points

Rotate given points based on a starting and ending vector.

skrobot.coordinates.geo.rotate points

 $skrobot.coordinates.geo.rotate_points(points, a, b)$

Rotate given points based on a starting and ending vector.

Axis vector k is calculated from the any two nonzero vectors a and b. Rotated points are calculated from following Rodrigues rotation formula.

$$P_{rot} = P\cos\theta + (k \times P)\sin\theta + k(k \cdot P)(1 - \cos\theta)$$

Parameters

- **points** (*numpy.ndarray*) Input points. The shape should be (3,) or (N, 3).
- a (numpy.ndarray) nonzero vector.

• **b** (numpy.ndarray) – nonzero vector.

Returns points_rot – rotated points.

Return type numpy.ndarray

3.4 Interfaces

3.4.1 Pybullet Interface

You can use a Pybullet interface using skrobot.

```
skrobot.interfaces._pybullet.
PybulletRobotInterface
```

Pybullet Interface Class

skrobot.interfaces._pybullet.PybulletRobotInterface

Pybullet Interface Class

Parameters

- robot (skrobot.model.RobotModel) robot model
- **urdf_path** (*None or str*) urdf path. If this value is *None*, get *urdf_path* from *robot.urdf_path*.
- **use_fixed_base** (*bool*) If this value is *True*, robot in pybullet simulator will be fixed.
- **connect** (*int*) pybullet's connection mode. If you have already connected to pybullet physics server, specify the server id. The default value is 1 (pybullet.GUI).

Examples

```
>>> from skrobot.models import PR2
>>> from skrobot.interfaces import PybulletRobotInterface
>>> robot_model = PR2()
>>> interface = PybulletRobotInterface(robot_model)
```

If you have already connected to pybullet physics server

```
>>> import pybullet
>>> client_id = pybullet.connect(pybullet.GUI)
>>> robot_model = PR2()
>>> interface = PybulletRobotInterface(robot_model, connect=client_id)
```

3.4. Interfaces 119

Methods

T()

Return 4x4 homogeneous transformation matrix.

Returns matrix – homogeneous transformation matrix shape of (4, 4)

Return type numpy.ndarray

Examples

```
>>> from numpy import pi
>>> from skrobot.coordinates import make_coords
>>> c = make_coords()
>>> c.T()
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]
>>> c.translate([0.1, 0.2, 0.3])
>>> c.rotate(pi / 2.0, 'y')
array([[ 2.22044605e-16, 0.00000000e+00,
                                           1.00000000e+00,
         1.00000000e-01],
       [ 0.00000000e+00, 1.00000000e+00,
                                           0.00000000e+00.
         2.00000000e-01],
       [-1.000000000e+00, 0.00000000e+00,
                                           2.22044605e-16,
         3.00000000e-01],
       [ 0.00000000e+00, 0.00000000e+00,
                                           0.00000000e+00,
         1.00000000e+00]])
```

angle_vector(angle_vector=None, realtime_simulation=None)

Send a angle vector to pybullet's phsyics engine.

Parameters

- angle_vector (None or numpy.ndarray) angle vector. If None, send self.robot.angle_vector()
- **realtime_simulation** (*None or bool*) If this value is *True*, send angle_vector by pybullet.setJointMotorControl2.

Returns angle vector – return sent angle vector.

Return type numpy.ndarray

static available()

Check Pybullet is available.

Returns _available – If *False*, pybullet is not available.

Return type bool

axis(ax)

changed()

Return False

This is used for CascadedCoords compatibility

Returns False – always return False

Return type bool

coords()

Return a deep copy of the Coordinates.

copy()

Return a deep copy of the Coordinates.

copy_coords()

Return a deep copy of the Coordinates.

copy_worldcoords()

Return a deep copy of the Coordinates.

difference_position(coords, translation_axis=True)

Return differences in position of given coords.

Parameters

- coords (skrobot.coordinates.Coordinates) given coordinates
- translation_axis (str or bool or None (optional)) we can take 'x', 'y', 'z', 'xy', 'yz', 'zx', 'xx', 'yy', 'zz', True or False(None).

Returns dif_pos – difference position of self coordinates and coords considering translation_axis.

Return type numpy.ndarray

Examples

difference_rotation(coords, rotation_axis=True)

Return differences in rotation of given coords.

Parameters

- **coords** (skrobot.coordinates.Coordinates) given coordinates
- rotation_axis(str or bool or None (optional)) we can take 'x', 'y', 'z', 'xx', 'yy', 'zz', 'xm', 'ym', 'zm', 'xy', 'yx', 'yz', 'zy', 'zx', 'xz', True or False(None).

Returns dif_rot – difference rotation of self coordinates and coords considering rotation_axis.

Return type numpy.ndarray

3.4. Interfaces 121

```
>>> from numpy import pi
>>> from skrobot.coordinates import Coordinates
>>> from skrobot.coordinates.math import rpy_matrix
>>> coord1 = Coordinates()
>>> coord2 = Coordinates(rot=rpy_matrix(pi / 2.0, pi / 3.0, pi / 5.0))
>>> coord1.difference_rotation(coord2)
array([-0.32855112, 1.17434985, 1.05738936])
>>> coord1.difference_rotation(coord2, rotation_axis=False)
array([0, 0, 0])
>>> coord1.difference_rotation(coord2, rotation_axis='x')
                 , 1.36034952, 0.78539816])
array([0.
>>> coord1.difference_rotation(coord2, rotation_axis='y')
array([0.35398131, 0.
                            , 0.97442695])
>>> coord1.difference_rotation(coord2, rotation_axis='z')
array([-0.88435715, 0.74192175,
                                            1)
```

Using mirror option ['xm', 'ym', 'zm'], you can allow differences of mirror direction.

disable_hook()

get_transform()

Return Transform object

Returns transform – corrensponding Transform to this coordinates

Return type skrobot.coordinates.base.Transform

inverse_rotate_vector(v)

inverse_transform_vector(vec)

Transform vector in world coordinates to local coordinates

Parameters vec (numpy.ndarray) – 3d vector. We can take batch of vector like (batch_size, 3)

Returns transformed_point - transformed point

Return type numpy.ndarray

inverse_transformation(dest=None)

Return a invese transformation of this coordinate system.

Create a new coordinate with inverse transformation of this coordinate system.

$$\left(\begin{array}{cc} R^{-1} & -R^{-1}p \\ 0 & 1 \end{array}\right)$$

Parameters dest (*None or* skrobot.coordinates.Coordinates) — If dest is given, the result of transformation is in-placed to dest.

Returns dest – result of inverse transformation.

Return type *skrobot.coordinates.Coordinates*

load_bullet()

Load bullet configurations.

This function internally called.

move_coords(target_coords, local_coords)

Transform this coordinate so that local coords to target coords.

Parameters

- target_coords (skrobot.coordinates.Coordinates) target coords.
- local_coords (skrobot.coordinates.Coordinates) local coords to be aligned.

Returns self.worldcoords() – world coordinates.

Return type skrobot.coordinates.Coordinates

```
newcoords(c, pos=None)
```

Update of position and orientation.

orient_with_matrix(rotation_matrix, wrt='world')

Force update this coordinate system's rotation.

Parameters

- **rotation_matrix** (*numpy.ndarray*) 3x3 rotation matrix.
- wrt (str or skrobot.coordinates.Coordinates) reference coordinates.

```
parent_orientation(v, wrt)
```

```
rotate(theta, axis=None, wrt='local')
```

Rotate this robot by given theta and axis.

For more detail, please see docs of skrobot.coordinates.Coordinates.rotate. The difference between the rotate, this function internally call pybullet.resetBasePositionAndOrientation.

Parameters

- theta (float) radian
- wrt (string or skrobot.coordinates.Coordinates) –

rotate_vector(v)

Rotate 3-dimensional vector using rotation of this coordinate

Parameters v (numpy.ndarray) – vector shape of (3,)

 $Returns \ np.matmul(self.rotation, v) - {\it rotated \ } vector$

Return type numpy.ndarray

3.4. Interfaces 123

```
>>> from skrobot.coordinates import Coordinates
>>> from numpy import pi
>>> c = Coordinates().rotate(pi, 'z')
>>> c.rotate_vector([1, 2, 3])
array([-1., -2., 3.])
```

rotate_with_matrix(mat, wrt='local')

Rotate this coordinate by given rotation matrix.

This is a subroutine of self.rotate function.

Parameters

- mat (numpy.ndarray) rotation matrix shape of (3, 3)
- wrt (str or skrobot.coordinates.Coordinates) with respect to.

Returns self

Return type skrobot.coordinates.Coordinates

rpy_angle()

Return a pair of rpy angles of this coordinates.

```
Returns rpy_angle(self.rotation) – a pair of rpy angles. See also skrobot.coordinates.math.rpy_angle
```

Return type tuple(numpy.ndarray, numpy.ndarray)

Examples

```
>>> import numpy as np
>>> from skrobot.coordinates import Coordinates
>>> c = Coordinates().rotate(np.pi / 2.0, 'x').rotate(np.pi / 3.0, 'z')
>>> r.rpy_angle()
(array([ 3.84592537e-16, -1.04719755e+00, 1.57079633e+00]),
array([ 3.14159265, -2.0943951 , -1.57079633]))
```

sync()

Synchronize pybullet pose to robot_model.

transform(c, wrt='local')

Transform this coordinate by coords based on wrt

For more detail, please see docs of skrobot.coordinates.Coordinates.transform. The difference between the transform, this function internally call pybullet.resetBasePositionAndOrientation.

Parameters

- c (skrobot.coordinates.Coordinates) coordinate
- wrt (string or skrobot.coordinates.Coordinates) If wrt is 'local' or self, multiply c from the right. If wrt is 'world' or 'parent' or self.parent, transform c with respect to worldcoord. If wrt is Coordinates, transform c with respect to c.

transform_vector(v)

"Return vector represented at world frame.

Vector v given in the local coords is converted to world representation.

```
Parameters v (numpy.ndarray) – 3d vector. We can take batch of vector like (batch_size, 3)
```

Returns transformed_point – transformed point

```
Return type numpy.ndarray
```

```
transformation(c2, wrt='local')
```

```
translate(vec, wrt='local')
```

Translate robot in simulator.

For more detail, please see docs of skrobot.coordinates.Coordinates.translate. The difference between the translate, this function internally call pybullet.resetBasePositionAndOrientation.

Parameters

- **vec** (*list or np.ndarray*) shape of (3,) translation vector. unit is [m] order.
- wrt (string or Coordinates (optional)) translate with respect to wrt.

wait_interpolation(thresh=0.05, timeout=60.0)

Wait robot movement.

This function usually called after self.angle_vector. Wait while the robot joints are moving or until time of timeout. This function called internally pybullet.stepSimulation().

Parameters

- **thresh** (*float*) velocity threshold for detecting movement stop.
- **timeout** (*float*) maximum time of timeout.

worldcoords()

Return thisself

worldpos()

Return translation of this coordinate

See also skrobot.coordinates.Coordinates.translation

Returns self.translation – translation of this coordinate

Return type numpy.ndarray

worldrot()

Return rotation of this coordinate

See also skrobot.coordinates.Coordinates.rotation

Returns self.rotation – rotation matrix of this coordinate

Return type numpy.ndarray

```
__eq__(value, /)
Return self==value.
__ne__(value, /)
Return self!=value.
__lt__(value, /)
Return self<value.
__le__(value, /)
Return self<=value.
__gt__(value, /)
Return self>value.
```

3.4. Interfaces 125

```
__ge__(value,/)
     Return self>=value.
__mul__(other c)
     Return Transformed Coordinates.
     Note that this function creates new Coordinates and does not change translation and rotation, unlike trans-
     form function.
         Parameters other_c (skrobot.coordinates.Coordinates) - input coordinates.
         Returns out – transformed coordinates multiplied other_c from the right. T = T_{self}
             T_{\text{other}_c}.
         Return type skrobot.coordinates.Coordinates
__pow__(exponent)
     Return exponential homogeneous matrix.
     If exponent equals -1, return inverse transformation of this coords.
         Parameters exponent (numbers. Number) – exponent value. If exponent equals -1, return in-
             verse transformation of this coords. In current, support only -1 case.
         Returns out - output.
         Return type skrobot.coordinates.Coordinates
Attributes
dimension
     Return dimension of this coordinate
         Returns len(self.translation) – dimension of this coordinate
         Return type int
dual_quaternion
     Property of DualQuaternion
     Return DualQuaternion representation of this coordinate.
         Returns DualQuaternion - DualQuaternion representation of this coordinate
         Return type skrobot.coordinates.dual_quaternion.DualQuaternion
name
     Return this coordinate's name
         Returns self. name – name of this coordinate
         Return type str
pose
     Getter of Pose in pybullet phsyics simulator.
     Wrapper of pybullet.getBasePositionAndOrientation.
         Returns pose – pose of this robot in the phsyics simulator.
         Return type skrobot.coordinates.Coordinates
quaternion
     Property of quaternion
```

Returns $\mathbf{q} - [\mathbf{w}, \mathbf{x}, \mathbf{y}, \mathbf{z}]$ quaternion

Return type numpy.ndarray

Examples

```
>>> from numpy import pi
>>> from skrobot.coordinates import make_coords
>>> c = make_coords()
>>> c.quaternion
array([1., 0., 0., 0.])
>>> c.rotate(pi / 3, 'y').rotate(pi / 5, 'z')
>>> c.quaternion
array([0.8236391 , 0.1545085 , 0.47552826, 0.26761657])
```

rotation

Return rotation matrix of this coordinates.

Returns self._rotation – 3x3 rotation matrix

Return type numpy.ndarray

Examples

translation

Return translation of this coordinates.

Returns self._translation – vector shape of (3,). unit is [m]

Return type numpy.ndarray

Examples

```
>>> from skrobot.coordinates import Coordinates
>>> c = Coordinates()
>>> c.translation
array([0., 0., 0.])
>>> c.translate([0.1, 0.2, 0.3])
>>> c.translation
array([0.1, 0.2, 0.3])
```

3.4. Interfaces 127

x_axis

Return x axis vector of this coordinates.

Returns axis – x axis.

Return type numpy.ndarray

y_axis

Return y axis vector of this coordinates.

Returns axis – y axis.

Return type numpy.ndarray

z_axis

Return z axis vector of this coordinates.

Returns axis – z axis.

Return type numpy.ndarray

```
>>> from skrobot.models import PR2
>>> from skrobot.interfaces import PybulletRobotInterface
>>> import pybullet
>>> client_id = pybullet.connect(pybullet.GUI)
>>> robot_model = PR2()
>>> interface = PybulletRobotInterface(robot_model, connect=client_id)
>>> interface.angle_vector(robot_model.reset_pose())
```

3.5 Signed distance function (SDF)

3.5.1 SDF classes

skrobot.sdf.SignedDistanceFunction	A base class for signed distance functions (SDFs).
skrobot.sdf.UnionSDF	One can concat multiple SDFs sdf_list by using this
	class.
skrobot.sdf.BoxSDF	SDF for a box specified by <i>origin</i> and <i>width</i> .
skrobot.sdf.GridSDF	SDF using precopmuted signed distances for gridded
	points.
skrobot.sdf.CylinderSDF	SDF for a cylinder specified by origin, radius and
	height
skrobot.sdf.SphereSDF	SDF for a sphere specified by <i>origin</i> and <i>radius</i> .

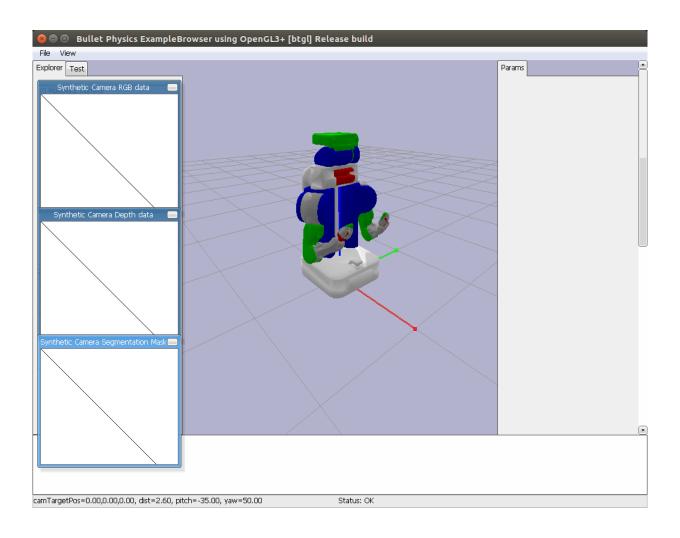
skrobot.sdf.SignedDistanceFunction

class skrobot.sdf.**SignedDistanceFunction**(origin, coords=None, use_abs=False)

A base class for signed distance functions (SDFs).

Suffixes _obj and _sdf (e.g. in points_sdf) indicate that points are expressed in the sdf's object coordinate (self.coords) and sdf-specific coordinate respectively. Each SDF performs the signed-distance computation in its own sdf-specific coordinate. For example, the origin of GridSDF's sdf-specific coordinate is the tip of the precomputed-gridded-box.

Usually, a child class implements the computation in the sdf-specific coordinates and SignedDistanceFunction wraps them so that the user can pass and get points and values expressed in an object coordinate. Thus, it is less



likely that a user directly calls a method in a child class.

Methods

```
__call__(points_obj)
```

Compute signed distances of input points to the implicit surface.

Parameters points_obj (numpy.ndarray[float](n_point, 3)) - 2 dim point array w.r.t. object coordinate.

Returns signed_distances – 1 dim (n_point,) array of signed distance.

Return type numpy.ndarray[float]

on_surface(points_obj)

Check if points are on the surface.

Parameters points_obj (numpy.ndarray[float](n_point, 3)) - 2 dim point array w.r.t. an object coordinate.

Returns

- **logicals** (*numpy.ndarray*[*bool*](*n_point*,)) boolean vector of the on-surface predicate for *points_obj*.
- **sd_vals** (*numpy.ndarray*[*float*](*n_point*,)) signed distances corresponding to *logicals*.

surface_points(n sample=1000)

Sample points from the implicit surface of the sdf.

Parameters n_sample (*int*) – number of sample points.

Returns

- **points_obj** (numpy.ndarray[float](n_point, 3)) sampled points w.r.t object coordinate.
- **dists** (*numpy.ndarray*[*float*](*n_point*,)) signed distances corresponding to points_obj.

```
__eq__(value,/)
Return self==value.
__ne__(value,/)
Return self!=value.
__lt__(value,/)
Return self<value.
__le__(value,/)
Return self<=value.
__gt__(value,/)
Return self>value.
__ge__(value,/)
Return self>=value.
```

skrobot.sdf.UnionSDF

class skrobot.sdf.UnionSDF(sdf list, coords=None)

One can concat multiple SDFs sdf_list by using this class.

For consistency in the concatenation, it is required that the all SDFs to be concated are with *use_abs=False*.

Methods

```
__call__(points_obj)
```

Compute signed distances of input points to the implicit surface.

Parameters points_obj (numpy.ndarray[float](n_point, 3)) - 2 dim point array w.r.t. object coordinate.

Returns signed_distances – 1 dim (n_point,) array of signed distance.

Return type numpy.ndarray[float]

classmethod from_robot_model(robot_model, dim_grid=50)

Create union sdf from a robot model

Parameters robot_model (skrobot.model.RobotModel) – Using the links of the robot model this creates the UnionSDF instance.

Returns union_sdf – union sdf of robot_model

Return type skrobot.sdf.UnionSDF

on_surface(points_obj)

Check if points are on the surface.

Parameters points_obj (numpy.ndarray[float](n_point, 3)) - 2 dim point array w.r.t. an object coordinate.

Returns

- **logicals** (*numpy.ndarray[bool]*(*n_point*,)) boolean vector of the on-surface predicate for *points_obj*.
- sd_vals (numpy.ndarray[float](n_point,)) signed distances corresponding to logicals.

surface_points(n_sample=1000)

Sample points from the implicit surface of the sdf.

Parameters n_sample (*int*) – number of sample points.

Returns

- **points_obj** (*numpy.ndarray*[*float*](*n_point*, 3)) sampled points w.r.t object coordinate.
- **dists** (*numpy.ndarray*[*float*](*n_point*,)) signed distances corresponding to points_obj.

```
__eq__(value,/)
    Return self==value.

__ne__(value,/)
    Return self!=value.
__lt__(value,/)
    Return self<value.
__le__(value,/)
    Return self<=value.
```

```
__gt__(value,/)
           Return self>value.
     __ge__(value,/)
          Return self>=value.
skrobot.sdf.BoxSDF
class skrobot.sdf.BoxSDF(origin, width, coords=None, use_abs=False)
     SDF for a box specified by origin and width.
     Methods
     __call__(points obj)
           Compute signed distances of input points to the implicit surface.
               Parameters points_obj (numpy.ndarray[float](n_point, 3)) - 2 dim point array w.r.t.
                   object coordinate.
               Returns signed_distances – 1 dim (n_point,) array of signed distance.
               Return type numpy.ndarray[float]
     on_surface(points_obj)
           Check if points are on the surface.
               Parameters points_obj (numpy.ndarray[float](n_point, 3)) - 2 dim point array w.r.t.
                   an object coordinate.
               Returns
                   • logicals (numpy.ndarray[bool](n_point,)) – boolean vector of the on-surface predicate for
                     points obj.
                   • sd vals (numpy.ndarray[float](n point,)) – signed distances corresponding to logicals.
     surface_points(n_sample=1000)
           Sample points from the implicit surface of the sdf.
               Parameters n_sample (int) – number of sample points.
               Returns
                   • points_obj (numpy.ndarray[float](n_point, 3)) – sampled points w.r.t object coordinate.
```

• **dists** (*numpy.ndarray*[*float*](*n_point*,)) – signed distances corresponding to points_obj.

```
__eq__(value,/)
    Return self==value.
__ne__(value,/)
    Return self!=value.
__lt__(value,/)
    Return self<value.
__le__(value,/)
    Return self<=value.
__gt__(value,/)
```

Return self>value.

```
__ge__(value,/)
Return self>=value.
```

skrobot.sdf.GridSDF

class skrobot.sdf.**GridSDF**(*sdf_data*, *origin*, *resolution*, *fill_value=inf*, *coords=None*, *use_abs=False*) SDF using precopmuted signed distances for gridded points.

Methods

```
__call__(points_obj)
```

Compute signed distances of input points to the implicit surface.

Parameters points_obj (numpy.ndarray[float](n_point, 3)) - 2 dim point array w.r.t. object coordinate.

Returns signed_distances – 1 dim (n_point,) array of signed distance.

Return type numpy.ndarray[float]

static from_file(filepath)

Return GridSDF instance from a .sdf file.

Parameters filepath (str or pathlib.Path) - path of .sdf file

Returns sdf_instance – instance of sdf

Return type skrobot.esdf.GridSDF

static from_objfile(obj_filepath, dim_grid=100, padding_grid=5)

Return GridSDF instance from an .obj file.

In the initial call of this method for an .obj file, the pre-process takes some time to converting it to a .sdf file. However, because a cache of .sdf file is created in the initial call, there is no overhead from the next call for the same .obj file.

Parameters

- obj_filepath(str or pathlib.Path) path of objfile
- dim_grid (int) dim of sdf
- padding_grid (int) number of padding

Returns sdf_instance - instance of sdf

Return type skrobot.sdf.GridSDF

is_out_of_bounds(points_obj)

check if the the input points is out of bounds

This method checks if the the input points is out of bounds of RegularGridInterpolator.

Parameters points_obj (numpy.ndarray[float](n_points, 3)) - points w.r.t. object to be checked.

Returns is_out_arr – If points is out of the interpolator's boundary, the correspoinding element of is out arr is True

Return type numpy.ndarray[bool](n_points,)

on_surface(points_obj)

Check if points are on the surface.

Parameters points_obj (numpy.ndarray[float](n_point, 3)) - 2 dim point array w.r.t. an object coordinate.

Returns

- **logicals** (*numpy.ndarray*[*bool*](*n_point*,)) boolean vector of the on-surface predicate for *points_obj*.
- sd_vals (numpy.ndarray[float](n_point,)) signed distances corresponding to logicals.

surface_points(n sample=1000)

Sample points from the implicit surface of the sdf.

Parameters n_sample (*int*) – number of sample points.

Returns

- **points_obj** (numpy.ndarray[float](n_point, 3)) sampled points w.r.t object coordinate.
- **dists** (*numpy.ndarray*[*float*](*n_point*,)) signed distances corresponding to points_obj.

```
__eq__(value,/)
    Return self==value.
__ne__(value,/)
    Return self!=value.
__lt__(value,/)
    Return self<value.
__le__(value,/)
    Return self<=value.
__gt__(value,/)
    Return self>value.
__ge__(value,/)
    Return self>=value.
```

skrobot.sdf.CylinderSDF

```
class skrobot.sdf.CylinderSDF(origin, height, radius, coords=None, use_abs=False) SDF for a cylinder specified by origin, 'radius' and height
```

Methods

```
__call__(points_obj)
```

Compute signed distances of input points to the implicit surface.

Parameters points_obj (numpy.ndarray[float](n_point, 3)) - 2 dim point array w.r.t. object coordinate.

Returns signed_distances – 1 dim (n_point,) array of signed distance.

Return type numpy.ndarray[float]

```
on_surface(points obj)
```

Check if points are on the surface.

Parameters points_obj (numpy.ndarray[float](n_point, 3)) - 2 dim point array w.r.t. an object coordinate.

Returns

- **logicals** (*numpy.ndarray[bool]*(*n_point*,)) boolean vector of the on-surface predicate for *points_obj*.
- **sd_vals** (*numpy.ndarray*[*float*](*n_point*,)) signed distances corresponding to *logicals*.

surface_points(n sample=1000)

Sample points from the implicit surface of the sdf.

Parameters n_sample (*int*) – number of sample points.

Returns

- points_obj (numpy.ndarray[float](n_point, 3)) sampled points w.r.t object coordinate.
- **dists** (*numpy.ndarray*[*float*](*n_point*,)) signed distances corresponding to points_obj.

```
__eq__(value,/)
Return self==value.
__ne__(value,/)
Return self!=value.
__lt__(value,/)
Return self<value.
__le__(value,/)
Return self<=value.
__gt__(value,/)
Return self>value.
__ge__(value,/)
Return self>=value.
```

skrobot.sdf.SphereSDF

class skrobot.sdf.**SphereSDF**(*origin*, *radius*, *coords=None*, *use_abs=False*)
SDF for a sphere specified by *origin* and *radius*.

Methods

```
__call__(points obj)
```

Compute signed distances of input points to the implicit surface.

Parameters points_obj (numpy.ndarray[float](n_point, 3)) - 2 dim point array w.r.t. object coordinate.

Returns signed_distances – 1 dim (n_point,) array of signed distance.

Return type numpy.ndarray[float]

on_surface(points_obj)

Check if points are on the surface.

Parameters points_obj $(numpy.ndarray[float](n_point, 3)) - 2 \dim point array w.r.t.$ an object coordinate.

Returns

- **logicals** (*numpy.ndarray[bool]*(*n_point*,)) boolean vector of the on-surface predicate for *points_obj*.
- **sd_vals** (*numpy.ndarray*[*float*](*n_point*,)) signed distances corresponding to *logicals*.

surface_points(n_sample=1000)

Sample points from the implicit surface of the sdf.

Parameters n_sample (*int*) – number of sample points.

Returns

- **points_obj** (*numpy.ndarray*[*float*](*n_point*, 3)) sampled points w.r.t object coordinate.
- **dists** (*numpy.ndarray*[*float*](*n_point*,)) signed distances corresponding to points_obj.

```
__eq__(value,/)
Return self==value.
__ne__(value,/)
Return self!=value.
__lt__(value,/)
Return self<value.
__le__(value,/)
Return self<=value.
__gt__(value,/)
Return self>value.
__ge__(value,/)
Return self>=value.
```

3.5.2 SDF utilities

skrobot.sdf.signed_distance_function. trimesh2sdf	Convert trimesh to signed distance function.
skrobot.sdf.signed_distance_function. link2sdf	Convert Link to corresponding sdf

skrobot.sdf.signed distance function.trimesh2sdf

skrobot.sdf.signed_distance_function.trimesh2sdf(mesh, dim_grid=100, padding_grid=5) Convert trimesh to signed distance function.

Parameters

- mesh (trimesh.base.Trimesh) mesh object.
- **dim_grid** (*int*) dimension of the GridSDF. This value is used for a not primitive mesh.
- padding_grid (int) number of padding.

Returns sdf – converted signed distance function.

Return type *skrobot.sdf.SignedDistanceFunction*

skrobot.sdf.signed_distance_function.link2sdf

skrobot.sdf.signed_distance_function.link2sdf(link, urdf_path, dim_grid=30)
Convert Link to corresponding sdf

Parameters

- link (skrobot.model.Link) link object
- **urdf_path** (*str*) urdf path of the robot model that the link belongs to
- **dim_grid** (*int*) dimension of the GridSDF

Returns sdf – corresponding signed distance function to the link type. e.g. if Link has geometry of urdf.Box, then BoxSDF is created.

Return type skrobot.sdf.SignedDistanceFunction

3.6 Planning

3.6.1 Sdf-swept-sphere-based collision checker

skrobot.planner.SweptSphereSdfCollisionCheckeCollision checker between swept spheres and sdf

skrobot.planner.SweptSphereSdfCollisionChecker

class skrobot.planner.SweptSphereSdfCollisionChecker(sdf, robot_model)

Collision checker between swept spheres and sdf

Methods

add_coll_spheres_to_viewer(viewer)

Add collision sheres to viewer

Parameters viewer (skrobot.viewers._trimesh.TrimeshSceneViewer) - viewer

add_collision_link(coll link)

Add link for which collision with sdf is checked

The given *coll_link* will be approximated by swept-spheres and these spheres will be added to collision sphere's list.

Parameters coll_link (skrobot.model.Link) - link for which collision with sdf is checked

add_collision_links(coll_links)

Add links for which collisions with SDF is checked.

The given *coll_links* will be approximated by swept-spheres and these spheres will be added to collision sphere's list.

Parameters coll_links (*list[skrobot.model.Link]*) – link list for which collisions with sdf is checked.

collision_check()

Check collision between links and collision spheres.

3.6. Planning 137

Returns is_collision – *True* if a collision occurred between any pair of links and collision spheres and *False* otherwise.

Return type bool

compute_batch_sd_vals(joint_list, angle_vector_seq, with_base=False, with_jacobian=False)

Compute sd signed distances of collision spheres

This method is the core of this class. We assume that this method is mainly called from a trajecotyr optimizer or path planner.

Let n_{wp} be the number of way-points of a trajectory. Let n_f be the number of collision feature points on the robot links.

Let $f_{i,j}: \mathbb{R}^{n_{dof}} \ni q \mapsto x \in \mathbb{R}^3$ be the forward kinematics of collision features point j at waypoint i where q is the angle vector and n_{dof} is the dimension of the angle vector.

Let $c: \mathbb{R}^3 \ni x \mapsto sd \in \mathbb{R}$ be the signd distance function.

Then, this function is defined as $F: \mathbb{R}^{n_{wp}n_{dof}} \ni \xi \mapsto [[f_{1,1}(q_1),\ldots,f_{1,n_f}(q_1)]^T,\ldots,[f_{n_{wp},1}(q_{n_{wp}}),\ldots,f_{n_{wp},n_f}(q_{n_{wp}})]^T]^T \in \mathbb{R}^{n_{wp}n_f}$ where $\xi = [q_1^T,\ldots,q_{n_{wp}}^T]^T$ be the angle vector sequence of the trajectory. The corresponding jacobian is defined as $\frac{\partial F}{\partial \varepsilon}$.

Parameters

- joint_list (list[skrobot.model.Joint]) joint list to be set
- angle_vector_seq (numpy.ndarray[float](n_wp, n_dof)) angle vector sequence.
- with_base (bool) hoge
- with_jacobian (bool) if True, jacobian is commuted at the same time.

Returns

- **sd_vals** (*numpy.ndarray[float]*(*n_wp* * *n_feature*,)) signed distnaces for all feature points through the trajectory
- **sd_vals_jacobi** (numpy.ndarray[float](n_wp * n_feature, n_wp * n_dof)) jacobain of sd_vals with respect to DOF (i.e. n_wp * n_dof) of the trajectory

delete_coll_spheres_from_viewer(viewer)

Delete collision sheres from viewer

Parameters viewer (skrobot.viewers._trimesh.TrimeshSceneViewer) - viewer

update_color()

Update the color of links under collision

This method checks the collision between the collision spheres and registered sdf. If collision spheres are found to be under collision, the color of the spheres will be changed to *color_collision_sphere*.

Returns dists – array of the signed distances for each sphere against sdf.

Return type numpy.ndarray(n_sphere,)

```
__eq__(value,/)
    Return self==value.
__ne__(value,/)
    Return self!=value.
__lt__(value,/)
    Return self<value.
```

```
__le__(value,/)
    Return self<=value.
__gt__(value,/)
    Return self>value.
__ge__(value,/)
    Return self>=value.
```

Attributes

n feature

Return number of collision sphere.

Returns n_feature – number of collision spheres.

Return type int

3.6.2 SQP-based trajectory planner

skrobot.planner.sqp_plan_trajectory	Gradient based trajectory optimization using scipy's
	SLSQP.

skrobot.planner.sqp plan trajectory

skrobot.planner.sqp_plan_trajectory(collision_checker, av_start, av_goal, joint_list, n_wp, safety_margin=0.01, with_base=False, weights=None, initial_trajectory=None, slsqp_option=None)

Gradient based trajectory optimization using scipy's SLSQP.

Collision constraint is considered in an inequality constraits. Terminal constraint (start and end) is considered as an equality constraint.

Parameters

- av_start (numpy.ndarray(n_dof,)) joint angle vector at start point
- **joint_list** (list[skrobot.model.Link]) link list to be controlled (similar to inverse_kinematics function)
- **n_wp** (*int*) number of waypoints
- safety_margin (float) safety margin in collision checking
- with_base (bool) If with_base=False, n_dof is the number of joints n_joint, but if with_base=True, n_dof = len(joint_list) + 3.
- weights (numpy.ndarray(n_dof,) or None) cost to move of each joint. For example, if you set weights=numpy.ndarray([1.0, 0.1, 0.1]) for a 3 DOF manipulator, moving the first joint is with high cost compared to others. If set to None it's automatically determined.
- initial_trajectory (numpy.ndarray(n_wp, n_dof) or None) initial solution in the trajectory optimization specified by a angle vector sequence. If None, initial trajectory is automatically generated.
- **slsqp_option** (*dict or None*) option of slsqp. Please see *options* in https://docs.scipy. org/doc/scipy/reference/optimize.minimize-slsqp.html for the detail. If set to *None*, a default values is used.

3.6. Planning 139

Returns planned_trajectory – planned trajectory.

Return type numpy.ndarray(n_wp, n_dof)

3.6.3 Swept sphere generater

skrobot.planner.swept_sphere.	Compute swept spheres approximating a mesh
compute_swept_sphere	

skrobot.planner.swept sphere.compute swept sphere

skrobot.planner.swept_sphere.compute_swept_sphere(collision_mesh, n_sphere=None, tol=0.1)
Compute swept spheres approximating a mesh

Parameters

- collision_mesh (trimesh. Trimesh) mesh which swept spheres are computed for
- **n_sphere** (*int or None*) number of sphere to approximate the mesh. If it's set to *None*, the number of sphere is automatically determined.
- **tol** (*float*) tolerance determins how much mesh jutting-out from the swept-spheres are accepted. Let *max_jut* be the maximum jut-distance. Then the setting *tol* enforces *max_jut* / *radius* < *max_jut*. If some integer is set to *n_sphere*, *tol* does not affects the result.

Returns

- **centers_original_space** (*numpy.ndarray*[*float*](*n_sphere*, 3)) center of the approximating spheres in the space where the mesh vertices are defined.
- radius (*float*) radius of approximating sphers.

3.6.4 Planner utils

skrobot.planner.utils.scipinize	Scipinize a function returning both f and jac
skrobot.planner.utils.set_robot_config	A utility function for setting robot state
skrobot.planner.utils.get_robot_config	A utility function for getting robot state
skrobot.planner.utils.	Compute fk for multiple feature points
forward_kinematics_multi	

skrobot.planner.utils.scipinize

```
skrobot.planner.utils.scipinize(fun)
Scipinize a function returning both f and jac
```

For the detail this issue may help: https://github.com/scipy/scipy/issues/12692

Parameters fun (*function*) – function maps numpy.ndarray(n_dim,) to tuple[numpy.ndarray(m_dim,), numpy.ndarray(m_dim, n_dim)], where the returned tuples is composed of function value(vector) and the corresponding jacobian.

Returns

• fun_scipinized (function) – function maps numpy.ndarray(n_dim,) to a value

numpy.ndarray(m_dim,).

• **fun_scipinized_jac** (*function*) – function maps numpy.ndarray(n_dim,) to jacobian numpy.ndarray(m_dim, n_dim).

skrobot.planner.utils.set robot config

skrobot.planner.utils.set_robot_config(robot_model, joint_list, av, with_base=False)
A utility function for setting robot state

Parameters

- robot_model (skrobot.model.CascadedLink) robot model
- joint_list (list[skrobot.model.Joint]) joint list to be set
- av (numpy.ndarray[float](n_dof,)) angle vector which has n_dof dims.
- with_base (bool) If with_base=False, n_dof is the number of joints n_joint, but if with_base=True, n_dof = n_joint + 3.

skrobot.planner.utils.get robot config

skrobot.planner.utils.get_robot_config(robot_model, joint_list, with_base=False)
A utility function for getting robot state

Parameters

- robot_model (skrobot.model.CascadedLink) robot model
- joint_list (list[Joint]) joint list of which you want to know the angles
- with_base (bool) If set to *True*, base position is also computed.

Returns av_whole (or av_joint) – angle vector. If $with_base=False$, n_dof is the number of joints n_joint , but if $with_base=True$, $n_dof = n_joint + 3$.

Return type numpy.ndarray(n_dof,)

skrobot.planner.utils.forward kinematics multi

skrobot.planner.utils.forward_kinematics_multi(robot_model, joint_list, av, move_target_list, with_rot, with_base, with_jacobian)

Compute fk for multiple feature points

Parameters

- robot_model (skrobot.model.CascadedLink) robot model.
- joint_list (list[skrobot.model.Joint]) joint to be controlled
- av (numpy.ndarray(n_dof,)) angle vector.
- move_target_list (list[skrobot.coordinates.CascadedCoords]) the list has n_feature elements. Each element is the coordinate of the features points.
- with_rot (bool) If set to *True*, 7(3 + 4) dim pose-fk is also computed. Otherwise, 3 dim point-fk is computed.
- with_base (bool) If with_base=False, n_dof is the number of joints n_joint, but if with_base=True, n_dof = n_joint + 3.

3.6. Planning 141

Returns

- **pose_arr** (*numpy.ndarray*(*n_feature*, *dim_pose*)) array of pose of each feature points. *dim_pose*=7' *if* 'with_rot=True. Otherwise, *dim_pose*=3.
- **jac_arr** (*numpy.ndarray*(*n_feature*, *dim_pose*, *n_dof*)) array of jacobian of each feature points.

CHAPTER

FOUR

DEVELOPMENT GUIDE

Read this guide before doing development in skrobot.

4.1 Setting Up

To set up the tools you'll need for developing, you'll need to install skrobot in development mode. Start by installing the development dependencies:

```
git clone https://github.com/iory/scikit-robot.git
cd scikit-robot
pip install -e .
```

4.2 Running Code Style Checks

We follow PEP 8 and partially OpenStack Style Guidelines as basic style guidelines. Any contributions in terms of code are expected to follow these guidelines.

You can use the autopep8, isort and the flake8 commands to check whether or not your code follows the guidelines. In order to avoid confusion from using different tool versions, we pin the versions of those tools. Install them with the following command (from within the top directory of the Chainer repository):

```
$ pip install hacking pytest autopep8 isort
```

And check your code with:

```
$ autopep8 path/to/your/code.py
$ flake8 path/to/your/code.py
```

autopep8 can automatically correct Python code to conform to the PEP 8 style guide:

```
$ autopep8 --in-place path/to/your/code.py
```

isort can automatically correct import order:

```
$ cd scikit-robot && isort path/to/your/code.py
```

For more information, please see the flake8 documentation.

4.3 Running Tests

This project uses pytest, the standard Python testing framework. Their website has tons of useful details, but here are the basics.

To run the testing suite, simply navigate to the top-level folder in scikit-robot and run the following command:

```
pytest -v tests
```

You should see the testing suite run. There are a few useful command line options that are good to know:

- -s Shows the output of stdout. By default, this output is masked.
- --pdb Instead of crashing, opens a debugger at the first fault.
- --lf Instead of running all tests, just run the ones that failed last.
- --trace Open a debugger at the start of each test.

You can see all of the other command-line options here.

By default, pytest will look in the tests folder recursively. It will run any function that starts with test_ in any file that starts with test_. You can run pytest on a directory or on a particular file by specifying the file path:

pytest -v tests/skrobot_tests/coordinates_tests/test_math.py

4.4 Building Documentation

To build scikit-robot's documentation, go to the docs directory and run make with the appropriate target. For example,

cd docs/
make html

will generate HTML-based docs, which are probably the easiest to read. The resulting index page is at docs/build/html/index.html. If the docs get stale, just run make clean to remove all build files.

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

146

PYTHON MODULE INDEX

S

```
skrobot, 7
skrobot.coordinates.geo, 118
skrobot.coordinates.math, 93
skrobot.interfaces._pybullet, 119
skrobot.model, 39
skrobot.models, 52
```

148 Python Module Index

INDEX

Symbols	method), 31
add() (skrobot.coordinates.dual quaternion.DualOu	ateration () (skrobot.interfacespybullet.PybulletRobotInterface
method), 37	methoa), 123
add() (skrobot.coordinates.quaternion.Quaternion	ge() (skrobot.model.RobotModel method), 50
method), 31	ge() (skrobot.models.fetch.Fetch method), 63
call() (skrobot.sdf.BoxSDF method), 132	ge() (skrobot.models.kuka.Kuka method), 77
call() (skrobot.sdf.CylinderSDF method), 134	ge() (skrobot.models.pr2.PR2 method), 91
call() (skrobot.sdf.GridSDF method), 133	ge() (skrobot.planner.SweptSphereSdfCollisionChecker
call() (skrobot.sdf.SignedDistanceFunction	method), 139
method), 130	ge() (skrobot.sdf.BoxSDF method), 132
call() (skrobot.sdf.SphereSDF method), 135	ge() (skrobot.sdf.CylinderSDF method), 135
call() (skrobot.sdf.UnionSDF method), 131	ge() (skrobot.sdf.GridSDF method), 134
div() (skrobot.coordinates.quaternion.Quaternion	ge() (skrobot.sdf.SignedDistanceFunction method),
method), 31	130
eq() (skrobot.coordinates.CascadedCoords	ge() (skrobot.sdf.SphereSDF method), 136
method), 24	ge() (skrobot.sdf.UnionSDF method), 132
eq() (skrobot.coordinates.Coordinates method), 14	gt() (skrobot.coordinates.CascadedCoords
eq() (skrobot.coordinates.dual_quaternion.DualQua	ternion method), 24
method), 36	gt() (skrobot.coordinates.Coordinates method), 14
eq() (skrobot.coordinates.quaternion.Quaternion	gt() (skrobot.coordinates.dual_quaternion.DualQuaternion
method), 31	method), 37
eq() (skrobot.interfacespybullet.PybulletRobotInter	rfacegt() (skrobot.coordinates.quaternion.Quaternion
method), 125	method), 31
eq() (skrobot.model.RobotModel method), 49	gt() (skrobot.interfacespybullet.PybulletRobotInterface
eq() (skrobot.models.fetch.Fetch method), 62	method), 125
eq() (skrobot.models.kuka.Kuka method), 76	gt() (skrobot.model.RobotModel method), 49
an () (skrobot models nr? PR? method) 90	gt() (skrobot.models.fetch.Fetch method), 62
eq() (skrobot.nlanner.SweptSphereSdfCollisionCheck	ker_gt() (skrobot.models.kuka.Kuka method), 76
method), 138	gt() (skrobol.models.pr2.1 K2 melnod), 90
eq() (skrobot.sdf.BoxSDF method), 132	gt() (skrobot.planner.SweptSphereSdfCollisionChecker
eq() (skrobot.sdf.CylinderSDF method), 135	method), 139
eq() (skrobot.sdf.GridSDF method), 134	gt() (skrobot.sdf.BoxSDF method), 132
eq() (skrobot.sdf.SignedDistanceFunction method),	gt() (skrobot.sdf.CylinderSDF method), 135
130	gt() (skrobot.sdf.GridSDF method), 134
eq() (skrobot.sdf.SphereSDF method), 136	gt() (skrobot.sdf.SignedDistanceFunction method),
eq() (skrobot.sdf.UnionSDF method), 131	130
ge() (skrobot.coordinates.CascadedCoords	gt() (skrobot.sdf.SphereSDF method), 136
method), 24	gt() (skrobot.sdf.UnionSDF method), 131
ge() (skrobot.coordinates.Coordinates method), 14	le() (skrobot.coordinates.CascadedCoords
ge() (skrobot.coordinates.dual_quaternion.DualQua	ternion method), 24
method), 37	le() (skrobot.coordinates.Coordinates method), 14
ge() (skrobot.coordinates.quaternion.Quaternion	le() (skrobot.coordinates.dual_quaternion.DualQuaternion

method), 36	ne() (skrobot.coordinates.CascadedCoords
le() (skrobot.coordinates.quaternion.Quaternion	method), 24
method), 31	ne() (skrobot.coordinates.Coordinates method), 14
	rfac ene() (skrobot.coordinates.dual_quaternion.DualQuaternion
method), 125	method), 36
le() (skrobot.model.RobotModel method), 49	ne() (skrobot.coordinates.quaternion.Quaternion
le() (skrobot.models.fetch.Fetch method), 62	method), 31
le() (skrobot.models.kuka.Kuka method), 76	ne() (skrobot.interfacespybullet.PybulletRobotInterface
le() (skrobot.models.pr2.PR2 method), 90	method), 125
le() (skrobot.planner.SweptSphereSdfCollisionCheck	
method), 138	ne() (skrobot.models.fetch.Fetch method), 62
le() (skrobot.sdf.BoxSDF method), 132	ne() (skrobot.models.kuka.Kuka method), 76
le() (skrobot.sdf.CylinderSDF method), 135	ne() (skrobot.models.pr2.PR2 method), 90
le() (skrobot.sdf.GridSDF method), 134	ne() (skrobot.planner.SweptSphereSdfCollisionChecker
le() (skrobot.sdf.SignedDistanceFunction method),	method), 138
130	ne() (skrobot.sdf.BoxSDF method), 132
le() (skrobot.sdf.SphereSDF method), 136	ne() (skrobot.sdf.CylinderSDF method), 135
le() (skrobot.sdf.UnionSDF method), 131	ne() (skrobot.sdf.GridSDF method), 134
lt() (skrobot.coordinates.CascadedCoords	ne() (skrobot.sdf.SignedDistanceFunction method),
method), 24	130
lt() (skrobot.coordinates.Coordinates method), 14	ne() (skrobot.sdf.SphereSDF method), 136
lt() (skrobot.coordinates.dual_quaternion.DualQua	
method), 36	neg() (skrobot.coordinates.quaternion.Quaternion
lt() (skrobot.coordinates.quaternion.Quaternion	method), 31
method), 31	pow() (skrobot.coordinates.CascadedCoords
lt() (skrobot.interfacespybullet.PybulletRobotInter	
method), 125	pow() (skrobot.coordinates.Coordinates method), 14
lt() (skrobot.model.RobotModel method), 49	
lt() (skrobot.models.fetch.Fetch method), 62 lt() (skrobot.models.kuka.Kuka method), 76	pow() (skrobot.interfacespybullet.PybulletRobotInterface method), 126
lt() (skrobot.models.pr2.PR2 method), 90	pow() (skrobot.model.RobotModel method), 50
lt() (skrobot.planner.SweptSphereSdfCollisionCheco	
method), 138	pow() (skrobot.models.kuka.Kuka method), 77
lt() (skrobot.sdf.BoxSDF method), 132	pow() (skrobot.models.pr2.PR2 method), 91
lt() (skrobot.sdf.CylinderSDF method), 135	rmul() (skrobot.coordinates.dual_quaternion.DualQuaternion
lt() (skrobot.sdf.GridSDF method), 134	method), 37
lt() (skrobot.sdf.SignedDistanceFunction method),	rmul() (skrobot.coordinates.quaternion.Quaternion
130	method), 31
lt() (skrobot.sdf.SphereSDF method), 136	sub() (skrobot.coordinates.quaternion.Quaternion
lt() (skrobot.sdf.UnionSDF method), 131	method), 31
mul() (skrobot.coordinates.CascadedCoords	truediv() (skrobot.coordinates.quaternion.Quaternion
method), 24	method), 31
mul() (skrobot.coordinates.Coordinates method),	_check_valid_rotation() (in module
14	skrobot.coordinates.math), 96
mul() (skrobot.coordinates.dual_quaternion.DualQu	
method), 37	skrobot.coordinates.math), 96
mul() (skrobot.coordinates.quaternion.Quaternion	_wrap_axis() (in module skrobot.coordinates.math), 94
method), 31	_
mul() (skrobot.interfacespybullet.PybulletRobotInt	en fa ce
method), 126	add_coll_spheres_to_viewer()
mul() (skrobot.model.RobotModel method), 50	(skrobot.planner.SweptSphereSdfCollisionChecker
mul() (skrobot.models.fetch.Fetch method), 63	method), 137
mul() (skrobot.models.kuka.Kuka method), 77	add_collision_link()
mul() (skrobot.models.pr2.PR2 method), 91	(skrobot planner Swent Sphere Sdf Collision Checker

method), 137	<pre>calc_inverse_jacobian()</pre>
add_collision_links()	(skrobot.models.kuka.Kuka method), 68
	calc_inverse_jacobian() (skrobot.models.pr2.PR2
method), 137	method), 82
	<pre>@malc_inverse_kinematics_nspace_from_link_list()</pre>
attribute), 37	(skrobot.model.RobotModel method), 41
	<pre>calc_inverse_kinematics_nspace_from_link_list()</pre>
tribute), 32	(skrobot.models.fetch.Fetch method), 54
angle_between_vectors() (in module skrobot.coordinates.math), 107	<pre>calc_inverse_kinematics_nspace_from_link_list()</pre>
	boalmerjamerse_kinematics_nspace_from_link_list()
method), 120	(skrobot.models.pr2.PR2 method), 82
angle_vector() (skrobot.model.RobotModel method),	<pre>calc_inverse_kinematics_weight_from_link_list()</pre>
40	(skrobot.model.RobotModel method), 41
<pre>angle_vector() (skrobot.models.fetch.Fetch method),</pre>	<pre>calc_inverse_kinematics_weight_from_link_list()</pre>
53	(skrobot.models.fetch.Fetch method), 54
<pre>angle_vector() (skrobot.models.kuka.Kuka method),</pre>	<pre>calc_inverse_kinematics_weight_from_link_list()</pre>
67	(skrobot.models.kuka.Kuka method), 68
<pre>angle_vector() (skrobot.models.pr2.PR2 method), 81</pre>	<pre>calc_inverse_kinematics_weight_from_link_list()</pre>
<pre>assoc() (skrobot.coordinates.CascadedCoords method),</pre>	(skrobot.models.pr2.PR2 method), 82
17	<pre>calc_jacobian_for_interlocking_joints()</pre>
assoc() (skrobot.model.RobotModel method), 40	(skrobot.model.RobotModel method), 41
assoc() (skrobot.models.fetch.Fetch method), 53	<pre>calc_jacobian_for_interlocking_joints()</pre>
assoc() (skrobot.models.kuka.Kuka method), 67	(skrobot.models.fetch.Fetch method), 54
assoc() (skrobot.models.pr2.PR2 method), 81	<pre>calc_jacobian_for_interlocking_joints()</pre>
$available () {\it (skrobot.interfaces._pybullet.PybulletRobotInterfaces._pybulletRobotInterfaces._py$	
static method), 120	<pre>calc_jacobian_for_interlocking_joints()</pre>
$\verb"axis" (skrobot.coordinates.dual_quaternion. Dual Quaternion") and a substitution of the properties of the properties$	
attribute), 37	<pre>calc_jacobian_from_link_list()</pre>
axis (skrobot.coordinates.quaternion.Quaternion at-	(skrobot.model.RobotModel method), 41
tribute), 32	<pre>calc_jacobian_from_link_list()</pre>
<pre>axis() (skrobot.coordinates.CascadedCoords method),</pre>	(skrobot.models.fetch.Fetch method), 54
18	<pre>calc_jacobian_from_link_list()</pre>
axis() (skrobot.coordinates.Coordinates method), 8	(skrobot.models.kuka.Kuka method), 68
axis()(skrobot.interfacespybullet.PybulletRobotInterfac	
method), 120	(skrobot.models.pr2.PR2 method), 82
axis() (skrobot.model.RobotModel method), 41	calc_joint_angle_from_min_max_table()
axis() (skrobot.models.fetch.Fetch method), 54	(skrobot.model.RobotModel method), 41
axis() (skrobot.models.kuka.Kuka method), 68	calc_joint_angle_from_min_max_table()
axis() (skrobot.models.pr2.PR2 method), 82	(skrobot.models.fetch.Fetch method), 54
axis_angle_from_matrix() (in module	calc_joint_angle_from_min_max_table()
<pre>skrobot.coordinates.math), 106 axis_angle_from_quaternion() (in module</pre>	(skrobot.models.kuka.Kuka method), 68 calc_joint_angle_from_min_max_table()
axis_angle_from_quaternion() (in module skrobot.coordinates.math), 118	(skrobot.models.pr2.PR2 method), 82
skrobot.cooramates.main), 116	calc_joint_angle_speed()
В	(skrobot.model.RobotModel method), 41
	calc_joint_angle_speed()
BoxSDF (class in skrobot.sdf), 132	(skrobot.models.fetch.Fetch method), 54
C	calc_joint_angle_speed()
	(skrobot.models.kuka.Kuka method), 68
<pre>calc_inverse_jacobian() (skrobot.model.RobotModel method), 41</pre>	calc_joint_angle_speed() (skrobot.models.pr2.PR2
calc_inverse_jacobian()	method), 82
(skrobot.models.fetch.Fetch method), 54	<pre>calc_joint_angle_speed_gain()</pre>
(a	(skrobot.model.RobotModel method), 42

<pre>calc_joint_angle_speed_gain() (skrobot.models.fetch.Fetch method), 55</pre>	<pre>calc_vel_from_rot() (skrobot.model.RobotModel method), 42</pre>		
calc_joint_angle_speed_gain() (skrobot.models.kuka.Kuka method), 68	calc_vel_from_rot() (skrobot.models.fetch.Fetch method), 55		
calc_joint_angle_speed_gain()	calc_vel_from_rot() (skrobot.models.kuka.Kuka		
(skrobot.models.pr2.PR2 method), 82	method), 69		
<pre>calc_nspace_from_joint_limit()</pre>	calc_vel_from_rot() (skrobot.models.pr2.PR2		
(skrobot.model.RobotModel method), 42	method), 83		
<pre>calc_nspace_from_joint_limit()</pre>	calc_weight_from_joint_limit()		
(skrobot.models.fetch.Fetch method), 55	(skrobot.model.RobotModel method), 42		
calc_nspace_from_joint_limit()	calc_weight_from_joint_limit()		
(skrobot.models.kuka.Kuka method), 68	(skrobot.models.fetch.Fetch method), 55		
calc_nspace_from_joint_limit()	calc_weight_from_joint_limit()		
(skrobot.models.pr2.PR2 method), 82	(skrobot.models.kuka.Kuka method), 69		
<pre>calc_target_axis_dimension()</pre>	calc_weight_from_joint_limit()		
(skrobot.model.RobotModel method), 42	(skrobot.models.pr2.PR2 method), 83		
<pre>calc_target_axis_dimension()</pre>	CascadedCoords (class in skrobot.coordinates), 17		
(skrobot.models.fetch.Fetch method), 55	changed() (skrobot.coordinates.CascadedCoords		
<pre>calc_target_axis_dimension()</pre>	method), 18		
(skrobot.models.kuka.Kuka method), 68	changed() (skrobot.coordinates.Coordinates method), 8		
<pre>calc_target_axis_dimension()</pre>	changed() (skrobot.interfacespybullet.PybulletRobotInterface		
(skrobot.models.pr2.PR2 method), 82	method), 120		
<pre>calc_target_joint_dimension()</pre>	changed() (skrobot.model.RobotModel method), 42		
(skrobot.model.RobotModel method), 42	changed() (skrobot.models.fetch.Fetch method), 55		
<pre>calc_target_joint_dimension()</pre>	changed() (skrobot.models.kuka.Kuka method), 69		
(skrobot.models.fetch.Fetch method), 55	changed() (skrobot.models.pr2.PR2 method), 83		
<pre>calc_target_joint_dimension()</pre>	close_hand() (skrobot.models.kuka.Kuka method), 69		
(skrobot.models.kuka.Kuka method), 68	<pre>collision_avoidance_link_pair_from_link_list()</pre>		
<pre>calc_target_joint_dimension()</pre>	(skrobot.model.RobotModel method), 42		
(skrobot.models.pr2.PR2 method), 82	<pre>collision_avoidance_link_pair_from_link_list()</pre>		
<pre>calc_union_link_list() (skrobot.model.RobotModel</pre>	(skrobot.models.fetch.Fetch method), 55		
method), 42	<pre>collision_avoidance_link_pair_from_link_list()</pre>		
<pre>calc_union_link_list() (skrobot.models.fetch.Fetch</pre>	(skrobot.models.kuka.Kuka method), 69		
method), 55	<pre>collision_avoidance_link_pair_from_link_list()</pre>		
<pre>calc_union_link_list() (skrobot.models.kuka.Kuka</pre>	(skrobot.models.pr2.PR2 method), 83		
method), 69	collision_check() (skrobot.planner.SweptSphereSdfCollisionChecker		
<pre>calc_union_link_list() (skrobot.models.pr2.PR2</pre>	method), 137		
method), 83	<pre>compute_batch_sd_vals()</pre>		
<pre>calc_vel_for_interlocking_joints()</pre>	(skrobot.planner.SweptSphereSdfCollisionChecker		
(skrobot.model.RobotModel method), 42	method), 138		
<pre>calc_vel_for_interlocking_joints()</pre>	<pre>compute_qp_common() (skrobot.model.RobotModel</pre>		
(skrobot.models.fetch.Fetch method), 55	method), 42		
<pre>calc_vel_for_interlocking_joints()</pre>	compute_qp_common() (skrobot.models.fetch.Fetch		
(skrobot.models.kuka.Kuka method), 69	method), 55		
<pre>calc_vel_for_interlocking_joints()</pre>	compute_qp_common() (skrobot.models.kuka.Kuka		
(skrobot.models.pr2.PR2 method), 83	method), 69		
<pre>calc_vel_from_pos() (skrobot.model.RobotModel</pre>	compute_qp_common() (skrobot.models.pr2.PR2		
method), 42	method), 83		
calc_vel_from_pos() (skrobot.models.fetch.Fetch	compute_swept_sphere() (in module		
method), 55	skrobot.planner.swept_sphere), 140		
calc_vel_from_pos() (skrobot.models.kuka.Kuka	compute_velocity() (skrobot.model.RobotModel		
method), 69	method), 42		
calc_vel_from_pos() (skrobot.models.pr2.PR2	compute_velocity() (skrobot.models.fetch.Fetch		
method), 83	method), 55		

<pre>compute_velocity()</pre>	(skrobot.models.kuka.Kuka	copy_worldcoords() method), 69	(skrobot.models.kuka.Kuka
<pre>compute_velocity() method), 83</pre>	(skrobot.models.pr2.PR2	copy_worldcoords() method), 83	(skrobot.models.pr2.PR2
	inates.dual_quaternion.DualQu		skrobot.sdf), 134
attribute), 37	_	_	
	dinates.quaternion.Quaternion	D	
attribute), 32 Coordinates (class in skr	robot.coordinates), 7	<pre>default_urdf_path</pre>	(skrobot.models.fetch.Fetch at-
coordinates_distance	() (in module	default_urdf_path	(skrobot.models.kuka.Kuka at-
	t.coordinates.CascadedCoords	tribute), 77 default_urdf_path	(skrobot.models.pr2.PR2 at-
* * * * * * * * * * * * * * * * * * * *	nates.Coordinates method), 8	tribute), 91	a from viousal)
	cespybullet.PybulletRobotInter	delete_coll_sphere	s_from_viewer() ner.SweptSphereSdfCollisionChecker
method), 121		method), 138	
coords() (skrobot.model.			.coordinates.CascadedCoords at-
coords() (skrobot.models		tribute), 25	
<pre>coords() (skrobot.models coords() (skrobot.models</pre>			model.RobotModel attribute), 50
	tes.CascadedCoords method),		models.fetch.Fetch attribute), 63
18	ies. euseudeu Coorus memou),		models.kuka.Kuka attribute), 77
copy() (skrobot.coordinat	tes.Coordinates method). 8	difference_position	models.pr2.PR2 attribute), 91
	es.dual_quaternion.DualQuater	-	dinates.CascadedCoords
method), 35	~	method), 18	unutes. Cuscute Coorus
copy() (skrobot.coor	dinates.quaternion.Quaternion	difference_position	n()
method), 31		_	dinates.Coordinates method),
	pybullet.PybulletRobotInterfac	ce 8	<i>,</i> ,
method), 121		difference_position	n()
copy() (skrobot.model.Ro		(skrobot.coord	$dinates.dual_quaternion.DualQuaternion$
copy() (skrobot.models.fe		method), 35	
copy() (skrobot.models.ki		difference_position	
<pre>copy() (skrobot.models.pr copy_coords() (skrobo</pre>	t.coordinates.CascadedCoords	(skrobot.inter) method), 121	facespybullet.PybulletRobotInterface
method), 18		difference_position	n() (skrobot.model.RobotModel
	robot.coordinates.Coordinates	method), 42	
	interfacespybullet.PybulletRob	<pre>difference_position potInterface_method), 55</pre>	n() (skrobot.models.fetch.Fetch
method), 121 copy_coords() (skrobot	t.model.RobotModel method),	<pre>difference_position method), 69</pre>	n() (skrobot.models.kuka.Kuka
42		difference_position	n() (skrobot.models.pr2.PR2
- -	models.fetch.Fetch method), 55	method), 83	*
= -	models.kuka.Kuka method), 69	difference_rotation	n()
	models.pr2.PR2 method), 83		dinates.CascadedCoords
	crobot.coordinates.CascadedCoo	,, 15	
method), 18	mahat aa andin ataa Caandin ataa	difference_rotation	
method), 8	crobot.coordinates.Coordinates	9	dinates.Coordinates method),
	crobot.interfacespybullet.Pybul	<i>lletRebelletterfase</i> otation	n()
method), 121			dinates.dual_quaternion.DualQuaternion
copy_worldcoords()	(skrobot.model.RobotModel	method), 36	
method), 42	(-ll	difference_rotation	
copy_worldcoords()	(skrobot.models.fetch.Fetch		facespybullet.PybulletRobotInterface
method), 55		method), 121	

<pre>difference_rotati method), 43</pre>		(skrobot.model.Ro	botModel		ntes.dual_quaternion), 35
difference_rotati	on()	(skrobot.models.fe	etch.Fetch	E	
method), 56				enforce_positive_q_re	ot_w()
difference_rotati		(skrobot.models.k	uka.Kuka	(skrobot.coordin	ates.dual_quaternion.DualQuaternion
method), 70				method), 36	
difference_rotati <i>method</i>), 84		(skrobot.models	s.pr2.PR2	F	
* * * * * * * * * * * * * * * * * * * *		oordinates.Cascad	edCoords		1.1.6.1) 50
attribute), 2		oramaies. Caseaa	cacooras	Fetch (class in skrobot.me	•
dimension (skrobot.		ates Coordinates (attribute)		it_weight_from_union_link_list()
15	coorain	aics.Coorainaics t	,,		RobotModel method), 44
dimension(skrobot.ii		espybullet.Pybull	etRobotInte	erface (skrobot.models.	<pre>it_weight_from_union_link_list() fetch.Fetch method), 57</pre>
attribute), 1		1 11 11 11 11			<pre>it_weight_from_union_link_list()</pre>
dimension (skrobot.n					kuka.Kuka method), 71
dimension (skrobot.n					<pre>it_weight_from_union_link_list()</pre>
dimension (skrobot.n				· · · · · · · · · · · · · · · · · · ·	pr2.PR2 method), 85
dimension (skrobot.n				find_link_path()	(skrobot.model.RobotModel
disable_hook() (sk		oordinates.Cascad	edCoords	method), 44	
method), 20				find_link_path()	(skrobot.models.fetch.Fetch
<pre>disable_hook()</pre>		bot.coordinates.Co	ordinates	method), 57	
method), 10				<pre>find_link_path()</pre>	(skrobot.models.kuka.Kuka
disable_hook() (ski	robot.in	terfacespybullet.	PybulletRo	botInterfacemethod), 71	
method), 12				<pre>find_link_path() (skreen)</pre>	obot.models.pr2.PR2 method),
<pre>disable_hook() (sk</pre>	krobot.n	nodel.RobotModel	method),	85	
44				<pre>find_link_route()</pre>	(skrobot.model.RobotModel
<pre>disable_hook() (sh</pre>	krobot.1	nodels.fetch.Fetch	method),	method), 44	
57 disable_hook() (sh	krobot.1	nodels.kuka.Kuka	method),	<pre>find_link_route() method), 57</pre>	(skrobot.models.fetch.Fetch
71				<pre>find_link_route()</pre>	(skrobot.models.kuka.Kuka
disable_hook() (sk		_		method), 71	
dissoc() (sk method), 20		oordinates.Cascad	edCoords	<pre>find_link_route() (skr 85</pre>	robot.models.pr2.PR2 method),
<pre>dissoc() (skrobot.me</pre>	odel.Ro	botModel method)	, 44	<pre>fix_leg_to_coords()</pre>	(skrobot.model.RobotModel
<pre>dissoc() (skrobot.me</pre>	odels.fe	tch.Fetch method),	57	method), 44	
<pre>dissoc() (skrobot.me</pre>	odels.kı	ıka.Kuka method),	71	<pre>fix_leg_to_coords()</pre>	(skrobot.models.fetch.Fetch
<pre>dissoc() (skrobot.me</pre>	odels.pi	2.PR2 method), 85	5	method), 57	`
dq (skrobot.coordinate attribute), 3'		_quaternion.DualQ	uaternion	<pre>fix_leg_to_coords() method), 71</pre>	(skrobot.models.kuka.Kuka
dual_quaternion(s	krobot.	coordinates.Cascad	dedCoords	fix_leg_to_coords()	(skrobot.models.pr2.PR2
attribute), 2	5			method), 85	(<i>o.u. ee e ie ue i.s.p.</i> 2 11 112
dual_quaternion (s tribute), 15		coordinates.Coord	inates at-	forward_kinematics_m	*
dual quaternion(s	krobota	nterfaces, pybulle	t.PvbulletR	skrobot.planner.i	f.GridSDF static method), 133
attribute), 1	26		,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	from chifile() (skrobol.sa)	of off CridSDE static method)
dual_quaternion		ot.model.RobotMo	del at-	133	ot.sdf.GridSDF static method),
<i>tribute</i>), 50	(3111 0 0				(skrobot.sdf.UnionSDF class
dual_quaternion <i>tribute</i>), 63	(skrol	oot.models.fetch.Fe	tch at-	method), 131	(skrobot.sdf.UnionSDF class
dual_quaternion	(skrob	ot.models.kuka.Ku	ıka at-	G	
tribute), 77				<pre>get_robot_config() (in</pre>	n module skrobot.planner.utils),
dual_quaternion (skrobot	models.pr2.PR2 d	attribute),	141	, , , , , , , , , , , , , , , , , , ,
91					ot.coordinates.CascadedCoords
DualQuaternion		(class	in	method), 20	

	skrobot.coordinates.Coordinates	<pre>inverse_kinematics_args()</pre>
method), 10		(skrobot.model.RobotModel method), 45
	$bot.interfaces._pybullet.PybulletR$	R <i>obnylencsfak</i> anematics_args()
method), 122		(skrobot.models.fetch.Fetch method), 58
<pre>get_transform() (skro.</pre>	bot.model.RobotModel method),	<pre>inverse_kinematics_args()</pre>
44		(skrobot.models.kuka.Kuka method), 72
<pre>get_transform() (skro</pre>	bot.models.fetch.Fetch method),	<pre>inverse_kinematics_args()</pre>
57		(skrobot.models.pr2.PR2 method), 86
<pre>get_transform() (skro</pre>	bot.models.kuka.Kuka method),	<pre>inverse_kinematics_loop()</pre>
71		(skrobot.model.RobotModel method), 45
<pre>get_transform() (skroa</pre>	bot.models.pr2.PR2 method), 85	<pre>inverse_kinematics_loop()</pre>
GridSDF (class in skrobo	et.sdf), 133	(skrobot.models.fetch.Fetch method), 58
<pre>gripper_distance()</pre>	(skrobot.models.pr2.PR2	<pre>inverse_kinematics_loop()</pre>
method), 85		(skrobot.models.kuka.Kuka method), 72
		<pre>inverse_kinematics_loop()</pre>
H		(skrobot.models.pr2.PR2 method), 86
head (skrobot.models.pr2	2.PR2 attribute), 91	<pre>inverse_kinematics_loop_for_look_at()</pre>
T	,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,	(skrobot.model.RobotModel method), 45
		<pre>inverse_kinematics_loop_for_look_at()</pre>
ik convergence check	k() (skrobot.model.RobotModel	(skrobot.models.fetch.Fetch method), 58
method), 44	x() (skroboi.modei.koboimodei	<pre>inverse_kinematics_loop_for_look_at()</pre>
	k() (skrobot.models.fetch.Fetch	(skrobot.models.kuka.Kuka method), 72
method), 57	x() (skrobot.modets.jetch.retch	<pre>inverse_kinematics_loop_for_look_at()</pre>
	k() (skrobot.models.kuka.Kuka	(skrobot.models.pr2.PR2 method), 86
method), 71	(SK10001.models.kuku.Kuku	<pre>inverse_kinematics_optimization()</pre>
ik_convergence_checl	k() (skrobot.models.pr2.PR2	(skrobot.model.RobotModel method), 45
method), 85	(SK10001.mouets.p12.1 K2	<pre>inverse_kinematics_optimization()</pre>
	nodel.RobotModel method), 45	(skrobot.models.fetch.Fetch method), 58
	nodels.fetch.Fetch method), 58	<pre>inverse_kinematics_optimization()</pre>
	nodels.kuka.Kuka method), 71	(skrobot.models.kuka.Kuka method), 72
	nodels.pr2.PR2 method), 86	<pre>inverse_kinematics_optimization()</pre>
interlocking_joint_p	=	(skrobot.models.pr2.PR2 method), 86
	RobotModel attribute), 50	inverse_rodrigues() (in module
interlocking_joint_p		skrobot.coordinates.math), 97
	s.fetch.Fetch attribute), 63	<pre>inverse_rotate_vector()</pre>
interlocking_joint_p		(skrobot.coordinates. Cascaded Coords
	s.kuka.Kuka attribute), 77	method), 20
	pairs (skrobot.models.pr2.PR2	<pre>inverse_rotate_vector()</pre>
attribute), 91	54115 (SW 0001oue151.p. 2.1 112	(skrobot.coordinates.Coordinates method),
	t.coordinates.dual_quaternion.Du	ualQuaternion
static method),		inverse_rotate_vector()
	nates.dual_quaternion.DualQuate	ernion (skrobot.interfacespybullet.PybulletRobotInterface
attribute), 37	icies.anai_quare/mon.2 uci guare	method), 122
	nates.quaternion.Quaternion at-	<pre>inverse_rotate_vector()</pre>
<i>tribute</i>), 32	teres.querermen.guerermen ar	(skrobot.model.RobotModel method), 45
inverse_kinematics()) (skrobot.model.RobotModel	<pre>inverse_rotate_vector()</pre>
method), 45	, (smootimedeliteoomiodel	(skrobot.models.fetch.Fetch method), 58
inverse_kinematics()) (skrobot.models.fetch.Fetch	<pre>inverse_rotate_vector()</pre>
method), 58	(SK10001.Models.jetem.i etch	(skrobot.models.kuka.Kuka method), 72
inverse_kinematics()) (skrobot.models.kuka.Kuka	<pre>inverse_rotate_vector() (skrobot.models.pr2.PR2</pre>
method), 71	, (S. C. C. C. MINOCO ESIMUMI I I I I I I I I I I I I I I I I I	method), 86
<pre>inverse_kinematics()</pre>	(skrobot.models.pr2.PR2	<pre>inverse_transform_vector()</pre>
method), 86	, (a	(skrobot.coordinates. Cascaded Coords
,, 00		method), 20

<pre>inverse_transform_vector()</pre>	L
(skrobot.coordinates.Coordinates method),	larm (skrobot.model.RobotModel attribute), 50
10	larm (skrobot.models.fetch.Fetch attribute), 63
<pre>inverse_transform_vector()</pre>	1 2rm (skrahot models kuka Kuka attributa) 77
(skrobot.interfacespybullet.PybulletRobotInterf	agam (skrobot, models, pr2.PR2 attribute), 91
method), 122	link2sdf() (in module
<pre>inverse_transform_vector()</pre>	skrobot.sdf.signed_distance_function), 137
(skrobot.model.RobotModel method), 45	link_lists() (skrobot.model.RobotModel method), 46
<pre>inverse_transform_vector()</pre>	link_lists() (skrobot.models.fetch.Fetch method), 59
(skrobot.models.fetch.Fetch method), 58	link_lists() (skrobot.models.kuka.Kuka method), 72
<pre>inverse_transform_vector()</pre>	link_lists() (skrobot.models.pr2.PR2 method), 87
(skrobot.models.kuka.Kuka method), 72	lleg (skrobot.model.RobotModel attribute), 50
<pre>inverse_transform_vector()</pre>	11eg (skrobot.models.fetch.Fetch attribute), 63
(skrobot.models.pr2.PR2 method), 86	11eg (skrobot.models.kuka.Kuka attribute), 77
<pre>inverse_transformation()</pre>	11eg (skrobot.models.pr2.PR2 attribute), 91
(skrobot.coordinates.CascadedCoords	load_bullet() (skrobot.interfacespybullet.PybulletRobotInterface
method), 20	method), 123
<pre>inverse_transformation()</pre>	load_urdf() (skrobot.model.RobotModel method), 46
(skrobot.coordinates.Coordinates method),	
10	load_urdf() (skrobot.models.fetch.Fetch method), 59
<pre>inverse_transformation()</pre>	load_urdf() (skrobot.models.kuka.Kuka method), 72
(skrobot.interfacespybullet.PybulletRobotInterf	load_urdf() (skrobot.models.pr2.PR2 method), 87 af6ad_urdf_file() (skrobot.model.RobotModel
method), 122	
<pre>inverse_transformation()</pre>	method), 46
(skrobot.model.RobotModel method), 45	load_urdf_file() (skrobot.models.fetch.Fetch
<pre>inverse_transformation()</pre>	method), 59
(skrobot.models.fetch.Fetch method), 58	load_urdf_file() (skrobot.models.kuka.Kuka
<pre>inverse_transformation()</pre>	method), 72
(skrobot.models.kuka.Kuka method), 72	<pre>load_urdf_file() (skrobot.models.pr2.PR2 method),</pre>
<pre>inverse_transformation() (skrobot.models.pr2.PR2</pre>	87
method), 86	look_at_hand() (skrobot.model.RobotModel method),
is_out_of_bounds() (skrobot.sdf.GridSDF method),	46
133	look_at_hand() (skrobot.models.fetch.Fetch method), 59
J	look_at_hand() (skrobot.models.kuka.Kuka method),
<pre>joint_max_angles (skrobot.model.RobotModel at-</pre>	72
tribute), 50	look_at_hand() (skrobot.models.pr2.PR2 method), 87
joint_max_angles (skrobot.models.fetch.Fetch at-	M
tribute), 63	
joint_max_angles (skrobot.models.kuka.Kuka at-	<pre>make_matrix() (in module skrobot.coordinates.math),</pre>
tribute), 77	97
<pre>joint_max_angles (skrobot.models.pr2.PR2 attribute),</pre>	manipulability() (in module
91	skrobot.coordinates.math), 108
joint_min_angles (skrobot.model.RobotModel at-	matrix2quaternion() (in module
tribute), 50	skrobot.coordinates.math), 114
joint_min_angles (skrobot.models.fetch.Fetch at-	matrix_exponent() (in module
tribute), 63	skrobot.coordinates.math), 103
joint_min_angles (skrobot.models.kuka.Kuka at-	<pre>matrix_log() (in module skrobot.coordinates.math),</pre>
tribute), 77	103
joint_min_angles (skrobot.models.pr2.PR2 attribute),	midcoords() (in module skrobot.coordinates.geo), 27
91	midpoint() (in module skrobot.coordinates.math), 99
71	midrot() (in module skrobot.coordinates.math), 99
K	module
	skrobot, 7
Kuka (class in skrobot.models.kuka), 65	skrobot.coordinates.geo,118

skrobot.coordinates.math, 93	newcoords() (skrobot.coordinates.CascadedCoords
skrobot.interfacespybullet,119	method), 21
skrobot.model, 39	newcoords() (skrobot.coordinates.Coordinates
skrobot.models,52	method), 10
move_coords() (skrobot.coordinates.CascadedCoords method), 20	newcoords() (skrobot.interfacespybullet.PybulletRobotInterface method), 123
<pre>move_coords() (skrobot.coordinates.Coordinates</pre>	newcoords() (skrobot.model.RobotModel method), 46
method), 10	newcoords() (skrobot.models.fetch.Fetch method), 59
<pre>move_coords() (skrobot.interfacespybullet.PybulletRob</pre>	onlemogords() (skrobot.models.kuka.Kuka method), 73
method), 123	newcoords() (skrobot.models.pr2.PR2 method), 87
<pre>move_coords() (skrobot.model.RobotModel method), 46</pre>	norm (skrobot.coordinates.dual_quaternion.DualQuaternion attribute), 37
<pre>move_coords() (skrobot.models.fetch.Fetch method), 59</pre>	norm (skrobot.coordinates.quaternion.Quaternion at-
<pre>move_coords() (skrobot.models.kuka.Kuka method), 72</pre>	tribute), 32
<pre>move_coords() (skrobot.models.pr2.PR2 method), 87</pre>	${\tt normalize()} \ (skrobot.coordinates.dual_quaternion.DualQuaternion$
<pre>move_end_pos() (skrobot.model.RobotModel method),</pre>	method), 36
46	normalize() (skrobot.coordinates.quaternion.Quaternion
<pre>move_end_pos() (skrobot.models.fetch.Fetch method),</pre>	method), 31
59	normalize_vector() (in module
<pre>move_end_pos() (skrobot.models.kuka.Kuka method),</pre>	skrobot.coordinates.math), 102
73	${\tt normalized} \ (skrobot.coordinates.dual_quaternion.DualQuaternion$
<pre>move_end_pos() (skrobot.models.pr2.PR2 method), 87</pre>	attribute), 38
<pre>move_end_rot() (skrobot.model.RobotModel method),</pre>	${\tt normalized} \ (skrobot. coordinates. quaternion. Quaternion$
46	attribute), 33
<pre>move_end_rot() (skrobot.models.fetch.Fetch method), 59</pre>	0
<pre>move_end_rot() (skrobot.models.kuka.Kuka method),</pre>	on_surface() (skrobot.sdf.BoxSDF method), 132
73	on_surface() (skrobot.sdf.CylinderSDF method), 134
<pre>move_end_rot() (skrobot.models.pr2.PR2 method), 87</pre>	on_surface() (skrobot.sdf.GridSDF method), 133
<pre>move_joints() (skrobot.model.RobotModel method),</pre>	on_surface() (skrobot.sdf.SignedDistanceFunction
46	method), 130
<pre>move_joints() (skrobot.models.fetch.Fetch method), 59</pre>	on_surface() (skrobot.sdf.SphereSDF method), 135
<pre>move_joints() (skrobot.models.kuka.Kuka method), 73</pre>	on_surface() (skrobot.sdf.UnionSDF method), 131
<pre>move_joints() (skrobot.models.pr2.PR2 method), 87</pre>	open_hand() (skrobot.models.kuka.Kuka method), 73
<pre>move_joints_avoidance()</pre>	orient_coords_to_axis() (in module
(skrobot.model.RobotModel method), 46	skrobot.coordinates.geo), 27
<pre>move_joints_avoidance()</pre>	<pre>orient_with_matrix()</pre>
(skrobot.models.fetch.Fetch method), 59	(skrobot.coordinates.CascadedCoords
<pre>move_joints_avoidance()</pre>	method), 21
(skrobot.models.kuka.Kuka method), 73	<pre>orient_with_matrix()</pre>
move_joints_avoidance() (skrobot.models.pr2.PR2 method), 87	(skrobot.coordinates.Coordinates method),
mentou), or	<pre>11 orient_with_matrix()</pre>
N	(skrobot.interfacespybullet.PybulletRobotInterface
$\verb"n_feature" (skrobot.planner.SweptSphereSdfCollisionChecollisio$	
attribute), 139	orient_with_matrix() (skrobot.model.RobotModel
name (skrobot.coordinates.CascadedCoords attribute), 25	method), 46
name (skrobot.coordinates.Coordinates attribute), 15	orient_with_matrix() (skrobot.models.fetch.Fetch
name (skrobot.interfacespybullet.PybulletRobotInterface	method), 59
attribute), 126	orient_with_matrix() (skrobot.models.kuka.Kuka
name (skrobot.model.RobotModel attribute), 50	method), 73
name (skrobot.models.fetch.Fetch attribute), 63	orient_with_matrix() (skrobot.models.pr2.PR2
name (skrobot.models.kuka.Kuka attribute), 77	method), 87
name (skrobot.models.pr2.PR2 attribute), 92	<i>"</i>

outer_product_matrix() (in module skrobot.coordinates.math), 104	<pre>quat_from_rpy() (in module</pre>
P	Quaternion (class in skrobot.coordinates.quaternion), 30
parent (skrobot.coordinates.CascadedCoords attribute), 25	quaternion (skrobot.coordinates.CascadedCoords attribute), 25
parent (skrobot.model.RobotModel attribute), 51	quaternion (skrobot.coordinates.Coordinates attribute),
parent (skrobot.models.fetch.Fetch attribute), 64 parent (skrobot.models.kuka.Kuka attribute), 78	${\tt quaternion} \ (skrobot.coordinates.dual_quaternion.DualQuaternion$
<pre>parent (skrobot.models.pr2.PR2 attribute), 92 parent_orientation()</pre>	attribute), 38 quaternion(skrobot.interfacespybullet.PybulletRobotInterface
(skrobot.coordinates.CascadedCoords	attribute), 126
method), 21	quaternion (skrobot.model.RobotModel attribute), 51
parent_orientation()	quaternion (skrobot.models.fetch.Fetch attribute), 64
(skrobot.coordinates.Coordinates method),	quaternion (skrobot.models.kuka.Kuka attribute), 78
11	quaternion (skrobot.models.pr2.PR2 attribute), 92
parent_orientation()	quaternion2matrix() (in module
(skrobot.interfacespybullet.PybulletRobotInterf	ace skrobot.coordinates.math), 114
method), 123	quaternion2rpy() (in module
<pre>parent_orientation() (skrobot.model.RobotModel</pre>	skrobot.coordinates.math), 115
method), 46	quaternion_absolute_distance() (in module
<pre>parent_orientation() (skrobot.models.fetch.Fetch</pre>	skrobot.coordinates.math), 113
method), 59	quaternion_conjugate() (in module
<pre>parent_orientation() (skrobot.models.kuka.Kuka</pre>	skrobot.coordinates.math), 111
method), 73	quaternion_distance() (in module
<pre>parent_orientation() (skrobot.models.pr2.PR2</pre>	skrobot.coordinates.math), 112
method), 87	quaternion_from_axis_angle() (in module
<pre>parentcoords() (skrobot.coordinates.CascadedCoords</pre>	skrobot.coordinates.math), 117
method), 21	quaternion_inverse() (in module
parentcoords() (skrobot.model.RobotModel method),	skrobot.coordinates.math), 111
47	quaternion_multiply() (in module
parentcoords() (skrobot.models.fetch.Fetch method),	skrobot.coordinates.math), 110 quaternion_norm() (in module
60	skrobot.coordinates.math), 113
parentcoords() (skrobot.models.kuka.Kuka method),	quaternion_normalize() (in module
73	skrobot.coordinates.math), 114
parentcoords() (skrobot.models.pr2.PR2 method), 88	quaternion_slerp() (in module
pose (skrobot.interfacespybullet.PybulletRobotInterface attribute), 126	skrobot.coordinates.math), 111
pose() (skrobot.coordinates.dual_quaternion.DualQuater	
method), 36	"R"
PR2 (class in skrobot.models.pr2), 79	random_coords() (in module skrobot.coordinates.base),
PybulletRobotInterface (class in	28
skrobot.interfacespybullet), 119	random_quaternion() (in module
suroounierjacespyomiet), 119	skrobot.coordinates.math), 110
Q	random_rotation() (in module
q (skrobot.coordinates.quaternion.Quaternion attribute),	skrobot.coordinates.math), 98
((skrobol.coordinates.quaternion.Quaternion attribute), 33	random_translation() (in module
qd(skrobot.coordinates.dual_quaternion.DualQuaternion	skrobot.coordinates.math), 98
attribute), 38	rarm (skrobot.model.RobotModel attribute), 51
qr (skrobot.coordinates.dual_quaternion.DualQuaternion	rarm (skrobot.models.fetch.Fetch attribute), 64
attribute), 38	rarm (skrobot.models.kuka.Kuka attribute), 78
<pre>quat_from_rotation_matrix() (in module</pre>	rarm (skrobot.models.pr2.PR2 attribute), 92
skrobot.coordinates.math), 116	reset_joint_angle_limit_weight()
//	(skrobot.model.RobotModel method), 47

reset_joint_angle_limit_weight()	method), 22
(skrobot.models.fetch.Fetch method), 60	<pre>rotate_with_matrix()</pre>
reset_joint_angle_limit_weight()	(skrobot.coordinates.Coordinates method),
(skrobot.models.kuka.Kuka method), 73	12
<pre>reset_joint_angle_limit_weight()</pre>	rotate_with_matrix()
(skrobot.models.pr2.PR2 method), 88	$(skrobot.interfaces._pybullet.PybulletRobotInterface$
<pre>reset_manip_pose() (skrobot.model.RobotModel</pre>	method), 124
method), 47	<pre>rotate_with_matrix() (skrobot.model.RobotModel</pre>
reset_manip_pose() (skrobot.models.fetch.Fetch	method), 47
method), 60	rotate_with_matrix() (skrobot.models.fetch.Fetch
reset_manip_pose() (skrobot.models.kuka.Kuka	method), 60
method), 73	rotate_with_matrix() (skrobot.models.kuka.Kuka
reset_manip_pose() (skrobot.models.pr2.PR2	method), 74
method), 88	rotate_with_matrix() (skrobot.models.pr2.PR2
reset_pose() (skrobot.model.RobotModel method), 47	method), 88
reset_pose() (skrobot.models.fetch.Fetch method), 60	rotation (skrobot.coordinates.CascadedCoords at-
reset_pose() (skrobot.models.kuka.Kuka method), 73	tribute), 25
reset_pose() (skrobot.models.pr2.PR2 method), 88	rotation (skrobot.coordinates.Coordinates attribute),
rleg (skrobot.model.RobotModel attribute), 51	15
rleg (skrobot.models.fetch.Fetch attribute), 64	$\verb"rotation" (skrobot.coordinates.dual_quaternion. Dual Quaternion") \\$
rleg (skrobot.models.kuka.Kuka attribute), 78	attribute), 38
rleg (skrobot.models.pr2.PR2 attribute), 92	${\tt rotation} (\textit{skrobot.coordinates.quaternion.} \textit{Quaternion}$
RobotModel (class in skrobot.model), 39	attribute), 33
rodrigues() (in module skrobot.coordinates.math), 105	$\verb"rotation" (skrobot.interfaces._pybullet.PybulletRobotInterface")$
rotate() (skrobot.coordinates.CascadedCoords	attribute), 127
method), 21	rotation (skrobot.model.RobotModel attribute), 51
rotate() (skrobot.coordinates.Coordinates method), 11	rotation (skrobot.models.fetch.Fetch attribute), 64
$\verb"rotate()" (skrobot.interfaces._pybullet.PybulletRobotInterfaces._pybullet.Pybul$	
method), 123	rotation (skrobot.models.pr2.PR2 attribute), 92
<pre>rotate() (skrobot.model.RobotModel method), 47</pre>	rotation_angle() (in module
rotate() (skrobot.models.fetch.Fetch method), 60	skrobot.coordinates.math), 97
rotate() (skrobot.models.kuka.Kuka method), 73	rotation_distance() (in module
rotate() (skrobot.models.pr2.PR2 method), 88	skrobot.coordinates.math), 106
rotate_matrix() (in module	rotation_matrix() (in module
skrobot.coordinates.math), 101	skrobot.coordinates.math), 100
<pre>rotate_points() (in module skrobot.coordinates.geo),</pre>	rotation_matrix_from_axis() (in module
118	skrobot.coordinates.math), 105
rotate_vector() (in module	rotation_matrix_from_quat() (in module
skrobot.coordinates.math), 100	skrobot.coordinates.math), 117
$\verb"rotate_vector"() \ (skrobot.coordinates. Cascaded Coords$	rotation_matrix_from_rpy() (in module
method), 21	skrobot.coordinates.math), 104
rotate_vector() (skrobot.coordinates.Coordinates	rpy2quaternion() (in module
method), 11	skrobot.coordinates.math), 115
$\verb"rotate_vector"() (skrobot.interfaces._pybullet.PybulletR")$	Po bpyl p anglee() (in module skrobot.coordinates.math), 102
method), 123	rpy_angle() (skrobot.coordinates.CascadedCoords
<pre>rotate_vector() (skrobot.model.RobotModel method),</pre>	method), 22
47	rpy_angle() (skrobot.coordinates.Coordinates
<pre>rotate_vector() (skrobot.models.fetch.Fetch method),</pre>	method), 12
60	$\verb"rpy_angle()" (skrobot.interfacespybullet.PybulletRobotInterface") and the property of th$
<pre>rotate_vector() (skrobot.models.kuka.Kuka method),</pre>	method), 124
74	<pre>rpy_angle() (skrobot.model.RobotModel method), 47</pre>
<pre>rotate_vector() (skrobot.models.pr2.PR2 method), 88</pre>	rpy_angle() (skrobot.models.fetch.Fetch method), 60
<pre>rotate_with_matrix()</pre>	rpy_angle() (skrobot.models.kuka.Kuka method), 74
(skrobot.coordinates.CascadedCoords	rpy_angle() (skrobot.models.pr2.PR2 method), 88

rpy_from_quat() (in mod skrobot.coordinates.math), 116	dule sync() (skrobot.interfacespybullet.PybulletRobotInterface method), 124
<pre>rpy_matrix() (in module skrobot.coordinates.ma 101</pre>	T
S	T() (skrobot.coordinates.CascadedCoords method), 17 T() (skrobot.coordinates.Coordinates method), 7 uaternin (skrobot.coordinates.dual_quaternion.DualQuaternion
attribute), 39 scipinize() (in module skrobot.planner.utils), 140	method), 35 T() (skrobot.coordinates.quaternion.Quaternion
$\verb screw_axis() (skrobot.coordinates.dual_quaternion) \\$	
<pre>method), 36 self_collision_check() (skrobot.model.RobotMo method), 48</pre>	<i>"</i>
self_collision_check() (skrobot.models.fetch.Fe	T() (skrobot.model.RobotModel method), 40 etch T() (skrobot.models.fetch.Fetch method), 53
method), 61	T() (skrobot.models.kuka.Kuka method), 65
self_collision_check() (skrobot.models.kuka.K	, (
<pre>method), 74 self_collision_check() (skrobot.models.pr2.l</pre>	transform() (in module skrobot.coordinates.math), 100 PR2 transform() (skrobot.coordinates.CascadedCoords
method), 89	method), 22
<pre>set_robot_config() (in module skrobot.planner.ut 141</pre>	
${\tt SignedDistanceFunction}\ ({\it class\ in\ skrobot.sdf}),\ 12$	transform() (skrobot.interfacespybullet.PybulletRobotInterface
skrobot	method), 124
<pre>module, 7 skrobot.coordinates.geo</pre>	transform() (skrobot.model.RobotModel method), 48
module, 118	transform() (skrobot.models.fetch.Fetch method), 61 transform() (skrobot.models.kuka.Kuka method), 75
skrobot.coordinates.math	transform() (skrobot.models.kukd.kukd method), 75 transform() (skrobot.models.pr2.PR2 method), 89
module, 93	transform_coords() (in module
skrobot.interfacespybullet	skrobot.coordinates.base), 29
module, 119	<pre>transform_vector() (skrobot.coordinates.CascadedCoords</pre>
skrobot.model	method), 22
module, 39	transform_vector()(skrobot.coordinates.Coordinates
skrobot.models	method), 13
module, 52 Sphere SPE (along in glaubet odf) 135	transform_vector() (skrobot.interfacespybullet.PybulletRobotInterfa
SphereSDF (class in skrobot.sdf), 135 sqp_plan_trajectory() (in module skrobot.plann	method), 124 er), transform_vector() (skrobot.model.RobotModel
139	method), 48
<pre>sr_inverse() (in module skrobot.coordinates.ma</pre>	
107	method), 61
<pre>sr_inverse_org()</pre>	
skrobot.coordinates.math), 108	method), 75
<pre>surface_points() (skrobot.sdf.BoxSDF method), 1</pre>	_ \ \
<pre>surface_points() (skrobot.sdf.CylinderSDF method</pre>	,, ,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
135	transformation() (skrobot.coordinates.CascadedCoords
<pre>surface_points() (skrobot.sdf.GridSDF method), surface_points() (skrobot.sdf.SignedDistanceFund</pre>	
method), 130	ction transformation() (skrobot.coordinates.Coordinates method), 13
surface_points() (skrobot.sdf.SphereSDF metho	
136	method), 125
<pre>surface_points() (skrobot.sdf.UnionSDF method</pre>	
131	method), 48
SweptSphereSdfCollisionChecker (class	<pre>in transformation() (skrobot.models.fetch.Fetch</pre>
skrobot.planner), 137	method), 61

transformation() <i>method</i>), 75	(skrobot.models.kuka.Kuka	<pre>worldcoords() (skrobot.model.RobotModel method), 49</pre>
transformation() (skrobot.models.pr2.PR2 method),		worldcoords() (skrobot.models.fetch.Fetch method), 62 worldcoords() (skrobot.models.kuka.Kuka method), 76
translate() (skrobe	ot.coordinates.CascadedCoords	worldcoords() (skrobot.models.pr2.PR2 method), 90
method), 23		worldpos() (skrobot.coordinates.CascadedCoords
* *	krobot.coordinates.Coordinates	method), 23
method), 13		worldpos() (skrobot.coordinates.Coordinates method),
translate()(skrobot.in	terfacespybullet.PybulletRobot	
method), 125		worldpos() (skrobot.interfacespybullet.PybulletRobotInterfac
<pre>translate() (skrobot.m</pre>	odel.RobotModel method), 48	method), 125
<pre>translate() (skrobot.m</pre>	odels.fetch.Fetch method), 61	<pre>worldpos() (skrobot.model.RobotModel method), 49</pre>
<pre>translate() (skrobot.m</pre>	odels.kuka.Kuka method), 75	worldpos() (skrobot.models.fetch.Fetch method), 62
<pre>translate() (skrobot.m</pre>	odels.pr2.PR2 method), 89	worldpos() (skrobot.models.kuka.Kuka method), 76
translation (skrobot.co	oordinates.CascadedCoords at-	<pre>worldpos() (skrobot.models.pr2.PR2 method), 90</pre>
tribute), 26		<pre>worldrot() (skrobot.coordinates.CascadedCoords</pre>
translation (skrobot	t.coordinates.Coordinates at-	method), 24
tribute), 16		<pre>worldrot() (skrobot.coordinates.Coordinates method),</pre>
	ordinates.dual_quaternion.Dual	Quaternion 14
attribute), 39		<pre>worldrot() (skrobot.interfacespybullet.PybulletRobotInterfac</pre>
	terfacespybullet.PybulletRobot	
attribute), 127		<pre>worldrot() (skrobot.model.RobotModel method), 49</pre>
	odel.RobotModel attribute), 51	worldrot() (skrobot.models.fetch.Fetch method), 62
	odels.fetch.Fetch attribute), 64	worldrot() (skrobot.models.kuka.Kuka method), 76
	odels.kuka.Kuka attribute), 78	worldrot() (skrobot.models.pr2.PR2 method), 90
	odels.pr2.PR2 attribute), 92	wxyz2xyzw() (in module skrobot.coordinates.math), 109
<pre>trimesh2sdf()</pre>	(in module ed_distance_function), 136	X
triple_product() skrobot.coordin	(in module ates.math), 96	x (skrobot.coordinates.quaternion.Quaternion attribute), 34
U		x_axis (skrobot.coordinates.CascadedCoords attribute),
UnionSDF (class in skrob	ot.sdf), 131	x_axis (skrobot.coordinates.Coordinates attribute), 16
	ot.coordinates.CascadedCoords	x_axis (skrobot.interfacespybullet.PybulletRobotInterface attribute), 127
update() (skrobot.model	RobotModel method) 49	x_axis (skrobot.model.RobotModel attribute), 52
update() (skrobot.model		x_axis (skrobot.models.fetch.Fetch attribute), 65
update() (skrobot.model		x_axis (skrobot.models.kuka.Kuka attribute), 79
update() (skrobot.models.pr2.PR2 method), 90		x_axis (skrobot.models.pr2.PR2 attribute), 93
		o xQh eck (sk robot.coordinates.quaternion.Quaternion at-
method), 138	The state of the s	tribute), 34
,,		xyzw2wxyz() (in module skrobot.coordinates.math), 109
W		, , , , , , , , , , , , , , , , , , , ,
w (skrobot.coordinates.quaternion.Quaternion attribute),		Y (classification and in atom superior fluctuation attailute)
34		y (skrobot.coordinates.quaternion.Quaternion attribute),
<pre>wait_interpolation()</pre>		34 farania (dual et condinates Caseaded Condo attribute)
method), 125	æspybutiet.FybutietRobotimerj	fayeaxis (skrobot.coordinates.CascadedCoords attribute), 26
	ot.coordinates.CascadedCoords	y_axis (skrobot.coordinates.Coordinates attribute), 16
method), 23		$\verb"y_axis" (skrobot.interfaces._pybullet.PybulletRobotInterface")$
worldcoords() (skrobot.coordinates.Coordinates		attribute), 128
method), 13		y_axis (skrobot.model.RobotModel attribute), 52
	.interfacespybullet.PybulletRol	poylyaexifac(eskrobot.models.fetch.Fetch attribute), 65
method) 125		v axis (skrobot models kuka Kuka attribute) 79

z_axis (skrobot.models.pr2.PR2 attribute), 93