SkyDy

Release 0.0.2

Scott M. Kyle

Jun 13, 2021

CONTENTS

A PDF version of these docs can be found here.

SkyDy is a Python packagel to provide a way to programmatically define an inter-connected mechnical system (ICMS). If you are new to SkyDy, start with some Examples.

This is the central page for all of SkyDy's documentation.

Contents:

CHAPTER

ONE

SKYDY PACKAGE

1.1 Subpackages

1.1.1 skydy.configuration package

Submodules

skydy.configuration.BaseSymbols module

class skydy.configuration.BaseSymbols.**BaseSymbols**(*name*, *identifier*, *is_coords=False*)

Bases: object

__init__(*name*, *identifier*, *is_coords=False*)

The base object for this whole package to function. It defines the sympy symbols that are used in the modelling process, whether it be translational (x,y,z), rotational (theta_x, theta_y, theta_z) coordinates, positions within a reference frame, or the direction and magnitude of forces and torques, all the symbols are generated by various combinations of this object.

Later, we have specific object thats handle these scenarios for us, but all inherit from this class. s such, we never explicility use this class

Parameters

- **name** (*int or str*) – the name for the symbols. This will form the superscript, i.e., the body the symbols refer to.
- **identifier** (*int or str*) – gives meaning to the symbol set as it identifies what or where the symbol set refer to. For a set of coordinates, this is "G", the centre of mass. For a force, torque or length it is F, tau and l respectively.
- **is_coords** (*int or str*) – defines if a sextuple is defines (representing the 6 coordinates) of a body, or a triple (representing the three Cartesian coordinates).

Example

A coordinate representation of body named "1", at its centre of mass, G.

```
>>> from skydy.configuration import BaseSymbols
>>> body_name = "1"
>>> body_loc = "G"
>>> body_symbols = BaseSymbols(body_name, body_loc, True)
```

A vector representation of force named "2"

```
>>> force_name = "2"
>>> force_id = "F"
>>> force_symbols = BaseSymbols(force_name, force_id)
```

A vector representation of distance, or dimension for a body named "1"

```
>>> dim_name = "1"
>>> dim_id = "1"
>>> dim_symbols = BaseSymbols(dim_name, dim_id)
```

as_dict()
Get the dictionary of symbol-value pairs.

assign_values(*values*, *idx=- 1*)
Assign value(s) to a symbol. By defualt, the value is instantiated as 1.

Parameters

- **values** (*array-like object, int or float*) – the values we want to assign. This must be an array-like object of the len(self.__symbols), or an integer. If an integer is provided, a valid index must be provided.
- **idx** (*int*) – for an int or float, the index to assign the value to. This must be a valid index in 0 to len(self.__symbols)-1.

Returns None

Example

```
>>> from skydy.configuration import BaseSymbols
>>> base_symbols = BaseSymbols("1", "G")
>>> # Assign all values
>>> base_symbols.assign_values([10, 20, 30, 40, 50, 60])
>>> # Assign just one value
>>> base_symbols.assign_values(24, 4)
```

sym_to_np(*sym_matrix*)
Convert a numeric sympy matrix, i.e., after substituting values, to a numpy matrix. We pass in the sympy matrix for flexbility down the line as to what we can and cannot convert.

Parameters **sym_matrix** (*sympy.matrices.dense.MutableDenseMatrix*) – a sympy matrix, with values substituted in.

Returns a numpy matrix of same dimensions of the input matrix.

Return type np_matrix (numpy.ndarray)

Example

```
>>> from skydy.configuration import BaseSymbols
>>> base_symbols = BaseSymbols("1", "G")
>>> # Assign all values
>>> base_symbols.assign_values([10, 20, 30, 40, 50, 60])
>>> # Crete a matrix of symbols
>>> sym_mat = base_symbols.symbols()
>>> # Substitute the values in
>>> sym_mat = sym_mat.subs(base_symbols.as_dict())
>>> # Convert to an numpy.ndarray.
>>> np_mat = base_symbols.sym_to_np(sym_mat)
```

symbols()
Get the sympy.matrices.dense.MutableDenseMatrix of the symbols.

values()
Return a numpy.ndarray of the values. If the values are not assigned, they return an array of ones by default.

class skydy.configuration.BaseSymbols.**CoordinateSymbols**(*name*)
Bases: *skydy.configuration.BaseSymbols.BaseSymbols*

__init__(*name*)
A set of coordinate symbols, inheriting from the BaseSymbols class. Coordinate symbols are simply the translational (x,y,z) and rotational (theta_x, theta_y, theta_z) coordinates about the centre of mass of a body.

As expected, use the RIGHT HAND RULE to determine coordinate directions.

Fix your x-axis appropriately, and the rest comes for free.

Parameters **name** (*int, float or string*) – the name of the body or object we want to generate coordinates for.

Returns None

Example

```
>>> from skydy.configuration import CoordinateSymbols
>>> # Create a set of symbols
>>> body_name = 1
>>> body_coords = CoordinateSymbols(body_name)
```

positions()
Returns the symbols of the positions (or coordinates) of the body.

velocities()
Returns the velocities of the positions (or coordinates) of the body.

class skydy.configuration.BaseSymbols.**DimensionSymbols**(*name*)
Bases: *skydy.configuration.BaseSymbols.BaseSymbols*

__init__(*name*)
A set of dimension symbols, inheriting from the BaseSymbols class. Dimension symbols simply exist in x-, y-, z-coordinates. By default, these symbols are defined by l, or a length.

Parameters **name** (*int, float or string*) – the name of the body or object we want to generate dimensions for.

Returns None

Example

```
>>> from skydy.configuration import DimensionSymbols
>>> # Create a set of symbols
>>> body_name = 1
>>> body_dims = DimensionSymbols(body_name)
```

class skydy.configuration.BaseSymbols.**ForceSymbols**(*name*)

Bases: *skydy.configuration.BaseSymbols.BaseSymbols*

__init__(*name*)

A set of force symbols, inheriting from the BaseSymbols class. Force symbols simply exist in x-, y-, z-coordinates. By default, these symbols are defined by F, or a Force.

Parameters **name** (*int, float or string*) – the name of the body or object we want to designate a force for.

Returns None

Example

```
>>> from skydy.configuration import ForceSymbols
>>> # Create a set of symbols
>>> body_name = 1
>>> body_forces = ForceSymbols(body_name)
```

class skydy.configuration.BaseSymbols.**TorqueSymbols**(*name*)

Bases: *skydy.configuration.BaseSymbols.BaseSymbols*

__init__(*name*)

A set of torque symbols, inheriting from the BaseSymbols class. Torque symbols simply exist as rotations about the x-, y-, z-coordinates. By default, these symbols are defined by tau, or a Torque.

Parameters **name** (*int, float or string*) – the name of the body or object we want to designate a Torque for.

Returns None

Example

```
>>> from skydy.configuration import TorqueSymbols
>>> # Create a set of symbols
>>> body_name = 1
>>> body_torques = TorqueSymbols(body_name)
```

skydy.configuration.Configuration module

class skydy.configuration.Configuration.**Configuration**(*name*)

Bases: *skydy.configuration.BaseSymbols.CoordinateSymbols*

__init__(*name*)

A body's configuration is nothing other than a description of its pose (where it is, and how it is oriented).

As such, it is decribed by a vector of positions, and a matrix of rotations. All bodies can have up to 6 DOFs, i.e., directions in which it can move.

By applying constraints, a body can have as little as zero DOFs.

A Configuration inherits from CoordinateSymbols, as it is solely related to a body's name, and the 6 CoordinateSymbols that describe it.

Parameters **name** (*int or str*) – the name for the symbols. This will form the superscript, i.e., the body the symbols refer to.

Returns None

Example

Configuration for body named "1".

```
>>> from skydy.configuration import Configuration
>>> body_name = "1"
>>> body_config = Configuration(body_name)
>>> # See the symbolic position and rotation of the body
>>> print(body_config.pos_body)
>>> print(body_config.rot_body)
```

accelerations()

Returns the acceleration of the coordinates of the body.

apply_constraint(*idx*, *const_value=0*)

Apply a coordinate constraint to the free configuration of the body. A constraint is nothing but a coordinate having constant value.

This indices for each coordinate are: 0 -> x 1 -> y 2 -> z 3 -> theta_x 4 -> theta_y 5 -> theta_z

Parameters

- **idx** (*int*) – the index to apply the constriaint to.
- **const_value** (*int or float*) – the constant value to substitute in for the coordinate at the index. Default value is zero.

Returns None

Example

Constrain some coordinate for a body named "1".

```
>>> from skydy.configuration import Configuration
>>> body_name = "1"
>>> body_config = Configuration(body_name)
>>> # Apply a translational constraint to the z-axis, at a height of 5 m.
>>> body_config.apply_constraint(2, 5)
>>> # Apply a rotational constraint about the y-axis
>>> body_config.apply_constraint(4, 0)
```

free_accelerations()
Return a list of free accelerations for the body.

free_coordinates()
Return a list of free coordinates for the body.

free_velocities()
Return a list of free velocities for the body.

property pos_body

reset_constraints()
Reset the constraints, i.e., remove any restrictions on movement.

In short, the position and rotation are the free configuration matrices determined on object instantiation.

property rot_body

Module contents

1.1.2 skydy.connectors package

Submodules

skydy.connectors.Connection module

class skydy.connectors.Connection.**Connection**(*body_in*, *joint*, *body_out*)
Bases: `object`

__init__(*body_in*, *joint*, *body_out*)
Define the connection fo two bodies, through a joint.

Parameters

- **body_in** (Body) – the input body
- **joint** (Joint) – the joint, defined as a common location for the input and output bodies, and the associated DOFs. Note, it is critical here, that the joint's input coordinate is in body_in coordinate frame, and the output coordinate is in body_out coordinate frame.
- **body_in** – the output body

Returns None

Examples

```
>>> from skydy.connectors import DOF, Connection, Joint
>>> from skydy.rigidbody import Body, BodyCoordinate, Ground
>>> # Two point-masses that meet at the origin
>>> p0 = BodyCoordinate("O")
>>> p1 = BodyCoordinate("G/O", 0, 0, 0)
>>> # Assume the joint can move in the x-coordinate
>>> j1 = Joint(p0, p1, [DOF(0)])
>>> # Define the two bodies
>>> b1 = Body()
>>> b2 = Body()
>>> # Define the connection
>>> cnx = Connection(b1, j1, b2)
```

as_dict()
Return a dictionary of the coordinate and properties of the connection.

property body_in

property body_out

draw(*ax=None*, *sub_vals=None*)
Draw the connection

Parameters

- **ax** (*matplotlib.axes._subplots.AxesSubplot*) – the axis to plot the connection on.
- **sub_vals** (*dict*) – symbol-value pairs required to go from symbolic to numeric expression. It is important to note, that all symbols each body is dependent on, for example, for upstream bodies and joints, are included.

Returns updated axes, with plots.

Return type ax (matplotlib.axes._subplots.AxesSubplot)

Example

```
>>> import matplotlib.pyplot as plt
>>> from skydy.connectors import DOF, Connection, Joint
>>> from skydy.rigidbody import Body, BodyCoordinate, Ground
```

```
>>> # Two point-masses that meet at the origin
>>> p0 = BodyCoordinate("O")
>>> p1 = BodyCoordinate("G/O", 0, 0, 0)
>>> # Assume the joint can move in the x-coordinate
>>> j1 = Joint(p0, p1, [DOF(0)], "J")
>>> # Define the two bodies
>>> b1 = Body()
>>> b2 = Body()
>>> # Define the connection
>>> cnx = Connection(b1, j1, b2)
>>> # Define the axes
>>> fig = plt.figure()
```

(continues on next page)

(continued from previous page)

```
>>> ax = fig.add_subplot(111, projection='3d')
>>> ax = cnx.draw(ax)
>>> plt.show()
```

global_configuration()

Propagate the configuration from the input body, to the output body through the joint.

Updates the attribute values in place.

property joint

skydy.connectors.DOF module

class skydy.connectors.DOF.**DOF**(*idx*, *free=True*, *const_value=0*)

Bases: object

__init__(*idx*, *free=True*, *const_value=0*)

A Degree of Freedom is nothing other than a body coordinate that is able to move.

Thus, to define a DOF, we need to simply supply the free index (idx). By default, if it is free, there is no constant value, so we do not need to supply the second, or third arguments.

Parameters

- **idx** (*int*) – the free coordinate index, between 0 and 5.
- **free** (*bool*) – if the coordinte at index idx is free. True by default.
- **const_value** (*int or float*) – If the DOF is not free, i.e., free=False on instantiation, we assign the constant value the coordinate has. By defualt this is zero.

Returns None

Example

Demonstrate all combinations of the DOF construction.

```
>>> from skydy.connectors import DOF
>>> # Define a DOF in the x-coordinate
>>> x_dof = DOF(0)
>>> # Note the following ALSO defines a coordinate in the y-direction
>>> y_dof = DOF(1, True)
>>> # Define a constraint in the z-direction, at a value of 10.
>>> z_con = DOF(1, False, 10)
>>> # Define a constraint in the theta_z-direction, at a value of 2.
>>> theta_z_con = DOF(5, False, 2)
```

property const_value

property free

property idx

skydy.connectors.Joint module

class skydy.connectors.Joint.**Joint**(*body_in_coord*, *body_out_coord*, *dof=None*, *name=None*)
Bases: object

__init__(*body_in_coord*, *body_out_coord*, *dof=None*, *name=None*)
A Joint a common location for two bodies to interact, and how the bodies can move relative to each other, based on the DOFs or constraints a joint has.

A Joint needs to be defined in the inputs AND output body's coordinate frames.

By default, a joint is assumed to be free in all directions. If a user defines a free or non-free DOF, we account for it. Any unspecified coordintes indices are assumed to be constrained at a value of zero.

Diagram:

z2 y2

z1 y1 | / | / p_j | / | / ____—->x<—-_____ |/ |/….—- ——O2——-x2 O1——-x1 ^G2 ^G1

Let Body 1 have CoM at O1, Body 2 have CoM at O2, and let a Joint exists at the point p_j. The point p_j can be described in both Body 1 and Body 2's coordinate frames. Note, the coordinate frames need not be aligned, or pointing in the same direction.

The body_in_coord, say this is Body 1, is the vector from 01 -> p_j in (x1, y1, z1). The body_out_coord, say this is Body 2, is the vector from 02 -> p_j in (x2, y2, z2).

The degrees of freedom are defined as motion in the input Body's coordinate frame.

Parameters

- **body_in_coord** (BodyCoordinate) – the location of the joint in the input body's coordinate frame.
- **body_out_coord** (BodyCoordinate) – the location of the joint in the output body's coordinate frame.
- **dof** (*list(DOF), or None*) – the list of DOFs for the joint. By default, all coordinates are free.
- **name** (*int or str*) – the name of the joint.

Returns None

Example

```
>>> from skydy.connectors import DOF
>>> from skydy.rigidbody import BodyCoordinate
>>> # Define the location of the joint in the input coordinate frame
>>> p_1 = BodyCoordinate(1, 10, 4, 5)
>>> # Define the location of the joint in the output coordinate frame
>>> p_2 = BodyCoordinate(2, -5, -3, -2)
>>> # Define the DOFs for our joint. Say x-direction and
>>> # theta_y directions
>>> j_dofs = [DOF(0), DOF(4)]
>>> # Define the joint
>>> joint = Joint(p_1, p_2, j_dofs, "J")
>>> # Check which DOFs are free. Expect 0 and 4.
>>> for dof in joint.dof:
```

(continues on next page)

(continued from previous page)

```
>>>     if dof.free:
>>>         print(dof.idx)
```

property body_in_coord

property body_out_coord

property dof

id_counter = 0

Module contents

1.1.3 skydy.inertia package

Submodules

skydy.inertia.InertiaMatrix module

class skydy.inertia.InertiaMatrix.**InertiaMatrix**(*name*)

Bases: object

__init__(*name*)

Define a generic inertia matrix about a body coordinates frame.

This is not a customizable oject. It is what it is, and it doesn't need to be any more.

Parameters **name** (*str, int*) – the body name for the inertia matrix

Returns None

Examples

```
>>> from skydy.inertia import InertiaMatrix
>>> i_mat = InertiaMatrix(1)
```

as_mat()

Return the inertia matrix as a sympy.matrix.

skydy.inertia.MassMatrix module

class skydy.inertia.MassMatrix.**MassMatrix**(*name*)

Bases: object

__init__(*name*)

Define a generic mass matrix about a body coordinates frame.

This is not a customizable oject. It is what it is, and it doesn't need to be any more.

Parameters **name** (*str, int*) – the body name for the mass matrix

Returns None

Examples

```
>>> from skydy.inertia import MassMatrix
>>> m_mat = MassMatrix(1)
```

as_mat()
Return the mass matrix as a sympy.matrix.

Module contents

1.1.4 skydy.multibody package

Submodules

skydy.multibody.MultiBody module

class skydy.multibody.MultiBody.**MultiBody**(*connections=None*, *name=None*)
Bases: `object`

__init__(*connections=None*, *name=None*)
A MultiBody is a sequence of Connections. If we are diligent with our definitions of coordinates, bodies and joints, the creation of MultiBody object is straightforward.

The most important Connection is the first one, as this is the connection that relates our MultiBody object to the Ground. All other Connections are then related to Body's that are defined in upstream, or earlier connections.

Parameters

- **connections** (*list(Connections)*) – a list of Connection objects. The first Connection in this list must reference the Ground.
- **name** (*str or int*) – the name of the MultiBody. Note, we do not allow duplicate MultiBody names.

Returns None

Example

Define a MultiBody with one connections.

```
>>> from skydy.rigidbody import Body, BodyCoordinate, BodyForce, BodyTorque,␣
↪Ground, GroundCoordinate
>>> from skydy.connectors import Connectors, DOF, Joint
>>> # Let's define a MultiBody (car) that moves in the x-direction only
>>> # Provide some dimensions
>>> l_car, w_car, h_car = 2, 1, 1
>>> car_name = "1"
>>> # Define the body
>>> b_car = Body(car_name)
>>> # Instantiate the car's dimensions
>>> b_car.dims.assign_values([l_car, w_car, h_car])
>>> # Add force to car in the car's x-coordinaate.
>>> F1 = BodyForce(name="1", x_dir=True)
```

(continues on next page)

(continued from previous page)

```
>>> # Force is applied at the COM
>>> force_loc = BodyCoordinate("PF1", 0, 0, 0)
>>> # Add the force at the location
>>> b_car.add_force(F1, force_loc)
>>> # Instantiate the ground
>>> b_gnd = Ground()
>>> p_gnd = GroundCoordinate()
>>> # Location of car wrt ground
>>> p_car = BodyCoordinate("G1/O", 0, 0, 0)
>>> # Degrees of freedom
>>> car_dofs = [DOF(0)]
>>> # Ground to car joint
>>> j1 = Joint(p_gnd, p_car, car_dofs, name=p_gnd.name)
>>> # The connection of the bodies through the joint
>>> cnx_car = Connection(b_gnd, j1, b_car)
>>> # The multibody object
>>> oned_car = MultiBody([cnx_car], "car")
```

add_connection(*connection*)
Add a connection to the MultiBody

Parameters **connection** (Connection) – a connection we want to to add.

Returns None

as_latex(*linearized=False*, *output_dir=None*, *file_name=None*, *include_diag=True*)
Generate the latex and pdf document deriving the equations of motion of the MultiBody object. Uses the skydy.output.LatexDocument object.

Parameters

- **linearized** (*bool*) – display the linear or nonlinear system matrices
- **output_dir** (*str or None*) – location to generate the .tex and .pdf outputs.
- **file_name** (*str or None*) – name for the .tex and .pdf files.
- **include_diag** (*bool*) – include the matplotlib generated diagram of the mutlibody.

Returns None.

property connections

draw(*output_dir=None*, *save_fig=True*)
Draw the MultiBody object. Uses the Body and Connection draw methods.

Parameters

- **output_dir** (*str or None*) – location to generate the .tex and .pdf outputs.
- **save_fig** (*bool*) – save or simply render the drawing.

eoms()
return the LHS and RHS of the equations of motion,

The LHS is the Riemannian Metric times the accelerations. The RHS is the Generalised Forces minus the potential functions, less any other dissipative forces.

force_symbols()
Returns the force symbols. We have to remove the coordinates and time symbols.

get_configuration()
Print the configuration, coordinates, dimensions of the bodies in the MultiBody object.

get_equilibria(*unforced=True*)
Get the equilibria configurations for the MultiBody.

Equilibria exist at configurations when the velocities and accelerations are zero. They can be forced or unforced.

For a system with EOMs of the form: x'' = f(x, x', u)

The unforced equlibria are the x0 such that f(x0, 0, 0) = 0.

The forced equilibria, x0, u0, satisfy f(x0, 0, u0) = 0.

Parameters **unforced** (*bool*) – returns the forced or unforced equilbria

Returns the coordinate equilibria values. force_eum (sympy.matrix): the force equilibria values

Return type coord_eum (sympy.matrix)

id_counter = 0

id_names = []

symbols()
Return the free symbols for the equations of motion.

system_matrices(*linearized=False*)
Returns the linear or non-linear system matrices.

Assume the MultiBody state x = (q, dq), where q are the generalised coordintae, and dq the associated velocities.

Then the system is described by:

M * x' = f(x, u),

where M is a block matrix with diagonal entries of the Identity and the Riemannian metric, i.e., M = blockdiag(I, G)

The linear system matrices are then defined by:

M * x' = A * x + B * u

where A = d/dx(f(x, u)), and B = d/du(f(x, u)).

To avoid overly complex expressions, we keep the M matrix on the LHS.

Parameters **linearized** (*bool*) – Return the linearized (or linear state-space) representation of the system, or nonlinear.

Returns the linear or nonlinear state transitions matrix. B (sympy.matrix)) the linear input matrix. If linearized=False, this is just the appropriately sized zero matrix.

Return type A (sympy.matrix)

skydy.multibody.MultiBody.**latexify**(*string_item*)
Recursive function to turn a string, or sympy.latex to a latex equation.

Parameters **string_item** (*str*) – string we want to make into an equation

Returns a latex-able equation.

Return type equation_item (str)

Module contents

1.1.5 skydy.output package

Submodules

skydy.output.Arrow3D module

class skydy.output.Arrow3D.**Arrow3D**(*xs*, *ys*, *zs*, **args*, ***kwargs*)
Bases: matplotlib.patches.FancyArrowPatch

__init__(*xs*, *ys*, *zs*, **args*, ***kwargs*)
Create a 3D Arrow.

draw(*renderer*)
Draw the Artist (and its children) using the given renderer.

This has no effect if the artist is not visible (*.Artist.get_visible* returns False).

Parameters **renderer** (*.RendererBase* subclass.) –

Notes

This method is overridden in the Artist subclasses.

skydy.output.LatexDocument module

class skydy.output.LatexDocument.**LatexDocument**
Bases: object

__init__()
A simple latex document that has sections and figures.

add_figure(*figure_name*, *file_name*)

add_section(*section_title*, *section_latex*)

del_section(*section_title*)

write_pdf(*file_name=None*, *output_dir=None*)

write_tex(*file_name=None*, *output_dir=None*)

skydy.output.get_output_dir module

skydy.output.get_output_dir.**get_output_dir**(*output_dir=None*)
Get the output directory, which is compatible with current directory.

Parameters **output_dir** (*str or None*) – the desired output directory

Returns the output directory, based on where function is called from.

Return type output_dir (str)

Module contents

1.1.6 skydy.rigidbody package

Submodules

skydy.rigidbody.Body module

class skydy.rigidbody.Body.**Body**(*name=None*)

Bases: *skydy.configuration.Configuration.Configuration*

__init__(*name=None*)

A Body is a collection of particles. It has a mass, and some dimensions (length, width and height), and by default has six degrees of freedom (DOFs). As such, it extends the Configuration class, with the addition of the inertial properties (InertiaMatrix and MassMatrix) and some dimensions (DimensionSymbols).

We also instantite two empty lists for the forces and torques that *might* be applied to the body.

Parameters **name** (*int, float or string*) – the name of the body.

Returns None

Example

```
>>> from skydy.rigidbody import Body
>>> # Empty initializer
>>> b_1 = Body()
>>> # Define name by a number
>>> b_2 = Body(2)
>>> # Empty initializer
>>> b_3 = Body("car")
```

add_force(*force_vector*, *force_location*)

Add, or apply a force to the body. For a force to do anything, the force must have a direction AND a location.

Parameters

- **force_vector** (BodyForce) – the force direction, as defined in the body's coordinate frame.
- **force_location** (BodyCoordinate) – the location of the force, as define in the body's coordinate frame.

Returns None

Examples

```
>>> from skydy.Body import Body, BodyForce, BodyCoordinate
>>> # Define the body
>>> b = Body()
>>> # Apply a force in the x-direction
>>> f_1 = BodyForce("1", x_dir=True)
>>> # Apply the force at the origin of the Body
>>> f_loc = BodyCoordinate("PF1")
```

(continues on next page)

(continued from previous page)

```
>>> # We can now apply the force
>>> b.apply_force(f_1, f_loc)
```

add_torque(*torque_vector*, *torque_location*)

Add, or apply a torque to the body. For a torque to f do anything, the torque must have a direction AND a location.

Parameters

- **torque_vector** (BodyTorque) – the torque direction, as defined in the body's coordinate frame.
- **torque_location** (BodyCoordinate) – the location of the torque, as define in the body's coordinate frame.

Returns None

Examples

```
>>> from skydy.Body import Body, BodyTorque, BodyCoordinate
>>> # Define the body
>>> b = Body()
>>> # Apply a torque in the x-direction
>>> t_1 = BodyTorque("1", x_dir=True)
>>> # Apply the torque at the origin of the Body
>>> t_loc = BodyCoordinate("PT1")
>>> # We can now apply the torque
>>> b.apply_torque(t_1, t_loc)
```

body_twists()

Calculate the body twists to ultimately determine the translational and rotational velocities.

omega is the body rotational velocity. v_body is the body translational velocity.

Mathematical expressions are:

Let r in R^3 be the position of the COM, and R in SO(3) the body orientation.

Then,

omega_hat = R^(-1) @ (d/dt) R, omega = unhat_map(omega_hat), v_body = (d/dt) r.

Refer to Bullo and Lewis for more information.

Parameters **None** –

Returns the body translational velocities omega_body (sympy.matrix): the body rotational velocities

Return type v_body (sympy.matrix)

draw(*ax=None*, *ref_body=None*, *ref_joint=None*, *sub_vals=None*)

Draw the body.

Parameters

- **ax** (*matplotlib.axes._subplots.AxesSubplot*) – the axis to plot the connection on.
- **ref_body** (*None or* Body) – the reference Body to propagate dimensions, coordinates etc.

- **ref_joint** (*None or numpy.ndarray*) – the location of the joint the body is connected to.
- **sub_vals** (*dict*) – symbol-value pairs required to go from symbolic to numeric expression. It is important to note, that all symbols each body is dependent on, for example, for upstream bodies and joints, are included.

Returns updated axes, with plots.

Return type ax (matplotlib.axes._subplots.AxesSubplot)

Example

```
>>> import matplotlib.pyplot as plt
>>> from skydy.rigidbody import Body
>>> # Define the body
>>> b = Body()
>>> # Define the axes
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111, projection='3d')
>>> ax = b.draw(ax)
>>> plt.show()
```

id_counter = 1

kinetic_energy()

Determine the kinetic energy of the body.

Let v_body in R^3, w_body in R^3 be the translational and rotational velocities, M in R^(3x3), I in R^(3x3) be the mass and inertia matrices, then,

KE_trans = (1/2) * v_vody^(T) @ M @ v_body KE_rot = (1/2) * w_vody^(T) @ I @ w_body

KE_tot = KE_trans + KE_rot.

Parameters **None** –

Returns the symbol expression of the kinetic energy.

Return type KE_tot (sympy.symbol)

potential_energy(*gravity*)

Determine the potential energy of the body.

The PE is the component of the position of the body in the z-coordinate (height), times gravity (g), times mass, i.e.,

PE = m * dot(g, r)

Parameters **gravity** (*sympy.matrix*) – the gravity vector in the global coordinte frame, typically [0, 0, g]

Returns the symbol expression of the potential energy

Return type potential_energy (sympy.symbols)

Examples

```
>>> from skydy.rigidbody import Body
>>> b = Body()
>>> # Define the gravity vector
>>> g = sym.Matrix([0, 0, sym.Symbol('g')])
>>> # Calculate the potential energy
>>> b.potential_energy(g)
```

class skydy.rigidbody.Body.**Ground**

Bases: *skydy.rigidbody.Body.Body*

__init__()

The base Body for every system.

The Ground defines the global coordinate frame, that cannot move or rotate. All coordinates are constrained, with zero constant value.

Parameters **None** –

Returns None

Example

```
>>> from skydy.rigidbody import Ground
>>> b_gnd = Ground()
```

skydy.rigidbody.Body.**symbols_to_latex**(*symbols*, *prefix=None*)

Convert symbols to latex.

Parameters

- **symbols** (*sympy.matrix or sympy.Symbol*) – a list of symbols or symbol.
- **prefix** (*str*) – an "equals" prefix.

Returns sympy object turned into a latex-able string.

Return type latex_str (str)

skydy.rigidbody.BodyCoordinate module

class skydy.rigidbody.BodyCoordinate.**BodyCoordinate**(*name*, *x=0*, *y=0*, *z=0*)

Bases: *skydy.configuration.BaseSymbols.DimensionSymbols*

__init__(*name*, *x=0*, *y=0*, *z=0*)

A coordinate of a point on a body. Inherits from DimensionSymbols.

Parameters

- **name** (*int, float or string*) – the name of the body or object we want to
- **dimensions for.** (*generate*) –
- **x** (*int or float*) – the x-coordinate to a point from the body's COM
- **y** (*int or float*) – the y-coordinate to a point from the body's COM
- **z** (*int or float*) – the z-coordinate to a point from the body's COM

Returns None

Example

```
>>> from skydy.configuration import BodyCoordinate
>>> # Create a set of symbols
>>> body_name = 1
>>> # Default constructor
>>> body_coord = BodyCoordinate(body_name)
>>> # Assign some distances
>>> body_coord = BodyCoordinate(body_name, 1, 2, 3)
>>> body_coord = BodyCoordinate(body_name, 3, -4, 4)
```

class skydy.rigidbody.BodyCoordinate.**GroundCoordinate**

Bases: *skydy.rigidbody.BodyCoordinate.BodyCoordinate*

__init__()

A BodyCoordinate with a default name of O, for origin, and (x, y, z) = (0, 0, 0)

Example

```
>>> from skydy.rigidbody import GroundCoordinate
>>> gnd_coord = GroundCoordinate()
```

skydy.rigidbody.BodyForce module

class skydy.rigidbody.BodyForce.**BodyForce**(*name*, *x_dir=False*, *y_dir=False*, *z_dir=False*)

Bases: *skydy.configuration.BaseSymbols.ForceSymbols*

__init__(*name*, *x_dir=False*, *y_dir=False*, *z_dir=False*)

Define a force that acts on a Body. The direction of the force is defined the the relevant body's coordinate frame.

By default, a force is assgined a default value of 1, if the force is defined in that direction, or 0 otherwise.

Parameters

- **name** (*int or str*) – the name for the force.
- **x_dir** (*bool*) – if the defined force acts in the body x-direction
- **y_dir** (*bool*) – if the defined force acts in the body y-direction
- **z_dir** (*bool*) – if the defined force acts in the body z-direction

Returns None

Example

```
>>> from skydy.rigidbody import BodyForce
>>> # Instantiate a force in the body x-direction
>>> f_1 = BodyForce("1", x_dir=True)
>>> # Instantiate another force in the body y-direction
>>> f_2 = BodyForce("2", y_dir=True)
>>> # Instantiate another force in all directions
>>> f_3 = BodyForce("3", True, True, True)
```

class skydy.rigidbody.BodyForce.**BodyTorque**(*name*, *x_dir=False*, *y_dir=False*, *z_dir=False*)

Bases: *skydy.configuration.BaseSymbols.TorqueSymbols*

__init__(*name*, *x_dir=False*, *y_dir=False*, *z_dir=False*)

Define a torque that acts on a Body. The direction of the torque is defined the the relevant body's coordinate frame, and a rotation ABOUT that coordinate direction (as per the right hand rule).

By default, a torque is assgined a default value of 1, if the torque is defined in that direction, or 0 otherwise.

Parameters

- **name** (*int or str*) – the name for the torque.
- **x_dir** (*bool*) – if the defined torque acts around the body x-direction
- **y_dir** (*bool*) – if the defined torque acts around the body y-direction
- **z_dir** (*bool*) – if the defined torque acts around the body z-direction

Returns None

Example

```
>>> from skydy.rigidbody import BodyTorque
>>> # Instantiate a torque about the body x-direction
>>> t_1 = BodyTorque("1", x_dir=True)
>>> # Instantiate another torque about the body y-direction
>>> t_2 = BodyTorque("2", y_dir=True)
>>> # Instantiate another torque about all directions
>>> t_3 = BodyTorque("3", True, True, True)
```

Module contents

1.2 Module contents

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

Symbols

A

B

C

D

E

F

G

I

J

K

L

M