DynamicistToolKit Documentation

Release 0.6.0.dev

Jason K. Moore

Aug 27, 2017

Contents

Contents:

CHAPTER 1

dtk Package

bicycle Module

inertia Module

dtk.inertia.**compound_pendulum_inertia**(*m*, *g*, *l*, *T*)

Returns the moment of inertia for an object hung as a compound pendulum.

Parameters **m** : float

Mass of the pendulum.

g : float

Acceration due to gravity.

l : float

Length of the pendulum.

T : float

The period of oscillation.

Returns **I** : float

Moment of interia of the pendulum.

dtk.inertia.**cylinder_inertia**(*l*, *m*, *ro*, *ri*)

Calculate the moment of inertia for a hollow cylinder (or solid cylinder) where the x axis is aligned with the cylinder's axis.

Parameters **l** : float

The length of the cylinder.

m : float

The mass of the cylinder.

ro : float

The outer radius of the cylinder.

ri : float

The inner radius of the cylinder. Set this to zero for a solid cylinder.

Returns **Ix** : float

Moment of inertia about cylinder axis.

Iy, Iz : float

Moment of inertia about cylinder axis.

`dtk.inertia.`**`euler_123`**(*angles*)

Returns the direction cosine matrix as a function of the Euler 123 angles.

Parameters **angles** : numpy.array or list or tuple, shape(3,)

Three angles (in units of radians) that specify the orientation of a new reference frame with respect to a fixed reference frame. The first angle, phi, is a rotation about the fixed frame's x-axis. The second angle, theta, is a rotation about the new y-axis (which is realized after the phi rotation). The third angle, psi, is a rotation about the new z-axis (which is realized after the theta rotation). Thus, all three angles are "relative" rotations with respect to the new frame. Note: if the rotations are viewed as occuring in the opposite direction (z, then y, then x), all three rotations are with respect to the initial fixed frame rather than "relative".

Returns **R** : numpy.matrix, shape(3,3)

Three dimensional rotation matrix about three different orthogonal axes.

`dtk.inertia.`**`euler_rotation`**(*angles*, *order*)

Returns a rotation matrix for a reference frame, B, in another reference frame, A, where the B frame is rotated relative to the A frame via body fixed rotations (Euler angles).

Parameters **angles** : array_like

An array of three angles in radians that are in order of rotation.

order : tuple

A three tuple containing a combination of `1`, `2`, and `3` where `1` is about the x axis of the first reference frame, `2` is about the y axis of the this new frame and `3` is about the z axis. Note that (1, 1, 1) is a valid entry and will give you correct results, but combinations like this are not necessarily useful for describing a general configuration.

Returns **R** : numpy.matrix, shape(3,3)

A rotation matrix.

Notes

The rotation matrix is defined such that a R times a vector v equals the vector expressed in the rotated reference frame.

v' = R * v

Where v is the vector expressed in the original reference frame and v' is the same vector expressed in the rotated reference frame.

Examples

```
>>> import numpy as np
>>> from dtk.inertia import euler_rotation
>>> angles = [np.pi, np.pi / 2., -np.pi / 4.]
>>> rotMat = euler_rotation(angles, (3, 1, 3))
>>> rotMat
matrix([[ -7.07106781e-01,   1.29893408e-16,  -7.07106781e-01],
        [ -7.07106781e-01,   4.32978028e-17,   7.07106781e-01],
        [  1.22464680e-16,   1.00000000e+00,   6.12323400e-17]])
>>> v = np.matrix([[1.], [0.], [0.]])
>>> vp = rotMat * v
>>> vp
matrix([[ -7.07106781e-01],
        [ -7.07106781e-01],
        [  1.22464680e-16]])
```

dtk.inertia.**inertia_components**(*jay*, *beta*)

Returns the 2D orthogonal inertia tensor.

When at least three moments of inertia and their axes orientations are known relative to a common inertial frame of a planar object, the orthoganal moments of inertia relative the frame are computed.

Parameters **jay** : ndarray, shape(n,)

An array of at least three moments of inertia. (n >= 3)

beta : ndarray, shape(n,)

An array of orientation angles corresponding to the moments of inertia in jay.

Returns **eye** : ndarray, shape(3,)

Ixx, Ixz, Izz

dtk.inertia.**parallel_axis**(*Ic*, *m*, *d*)

Returns the moment of inertia of a body about a different point.

Parameters **Ic** : ndarray, shape(3,3)

The moment of inertia about the center of mass of the body with respect to an orthogonal coordinate system.

m : float

The mass of the body.

d : ndarray, shape(3,)

The distances along the three ordinates that located the new point relative to the center of mass of the body.

Returns **I** : ndarray, shape(3,3)

The moment of inertia of a body about a point located by the distances in d.

dtk.inertia.**principal_axes**(*I*)

Returns the principal moments of inertia and the orientation.

Parameters **I** : ndarray, shape(3,3)

An inertia tensor.

Returns **Ip** : ndarray, shape(3,)

The principal moments of inertia. This is sorted smallest to largest.

C : ndarray, shape(3,3)

The rotation matrix.

dtk.inertia.**rotate3**(*angles*)

Produces a three-dimensional rotation matrix as rotations around the three cartesian axes.

Parameters **angles** : numpy.array or list or tuple, shape(3,)

Three angles (in units of radians) that specify the orientation of a new reference frame with respect to a fixed reference frame. The first angle is a pure rotation about the x-axis, the second about the y-axis, and the third about the z-axis. All rotations are with respect to the initial fixed frame, and they occur in the order x, then y, then z.

Returns **R** : numpy.matrix, shape(3,3)

Three dimensional rotation matrix about three different orthogonal axes.

dtk.inertia.**rotate3_inertia**(*RotMat*, *relInertia*)

Rotates an inertia tensor. A derivation of the formula in this function can be found in Crandall 1968, Dynamics of mechanical and electromechanical systems. This function only transforms an inertia tensor for rotations with respect to a fixed point. To translate an inertia tensor, one must use the parallel axis analogue for tensors. An inertia tensor contains both moments of inertia and products of inertia for a mass in a cartesian (xyz) frame.

Parameters **RotMat** : numpy.matrix, shape(3,3)

Three-dimensional rotation matrix specifying the coordinate frame that the input inertia tensor is in, with respect to a fixed coordinate system in which one desires to express the inertia tensor.

relInertia : numpy.matrix, shape(3,3)

Three-dimensional cartesian inertia tensor describing the inertia of a mass in a rotated coordinate frame.

Returns **Inertia** : numpy.matrix, shape(3,3)

Inertia tensor with respect to a fixed coordinate system ("unrotated").

dtk.inertia.**rotate_inertia_about_y**(*I*, *angle*)

Returns inertia tensor rotated through angle about the Y axis.

Parameters **I** : ndarray, shape(3,)

An inertia tensor.

angle : float

Angle in radians about the positive Y axis of which to rotate the inertia tensor.

dtk.inertia.**torsional_pendulum_inertia**(*k*, *T*)

Calculate the moment of inertia for an ideal torsional pendulum.

Parameters **k** : float

Torsional stiffness.

T : float

Period of oscillation.

Returns **I** : float

Moment of inertia.

dtk.inertia.**total_com**(*coordinates*, *masses*)
Returns the center of mass of a group of objects if the indivdual centers of mass and mass is provided.

coordinates [ndarray, shape(3,n)] The rows are the x, y and z coordinates, respectively and the columns are for each object.

masses [ndarray, shape(3,)] An array of the masses of multiple objects, the order should correspond to the columns of coordinates.

Returns **mT** : float

Total mass of the objects.

cT : ndarray, shape(3,)

The x, y, and z coordinates of the total center of mass.

dtk.inertia.**tube_inertia**(*l*, *m*, *ro*, *ri*)
Calculate the moment of inertia for a tube (or rod) where the x axis is aligned with the tube's axis.

Parameters **l** : float

The length of the tube.

m : float

The mass of the tube.

ro : float

The outer radius of the tube.

ri : float

The inner radius of the tube. Set this to zero if it is a rod instead of a tube.

Returns **Ix** : float

Moment of inertia about tube axis.

Iy, Iz : float

Moment of inertia about normal axis.

dtk.inertia.**x_rot**(*angle*)
Returns the rotation matrix for a reference frame rotated through an angle about the x axis.

Parameters **angle** : float

The angle in radians.

Returns **Rx** : np.matrix, shape(3,3)

The rotation matrix.

Notes

v' = Rx * v where v is the vector expressed the reference in the original reference frame and v' is the vector expressed in the new rotated reference frame.

dtk.inertia.**y_rot**(*angle*)
Returns the rotation matrix for a reference frame rotated through an angle about the y axis.

Parameters **angle** : float

The angle in radians.

Returns **Rx** : np.matrix, shape(3,3)

The rotation matrix.

Notes

v' = Rx * v where v is the vector expressed the reference in the original reference frame and v' is the vector expressed in the new rotated reference frame.

`dtk.inertia.`**`z_rot`**(*angle*)

Returns the rotation matrix for a reference frame rotated through an angle about the z axis.

Parameters **angle** : float

The angle in radians.

Returns **Rx** : np.matrix, shape(3,3)

The rotation matrix.

Notes

v' = Rx * v where v is the vector expressed the reference in the original reference frame and v' is the vector expressed in the new rotated reference frame.

`process` Module

CHAPTER 2

References

CHAPTER 3

Introduction

This is a collection of Python modules which contain tools that are helpful for a dynamicist. Right now it is basically a place I place general tools that don't necessarily need a distribution of their own.

CHAPTER 4

Modules

bicycle Generic tools for basic bicycle dynamics analysis.

inertia Various functions for calculating and manipulating inertial quantities.

process Various tools for common signal processing tasks.

CHAPTER 5

Installation

You will need Python 2.7 or 3.3+ and setuptools to install the packages. Its best to install the dependencies first (NumPy, SciPy, matplotlib, Pandas). The SciPy Stack instructions are helpful for this: http://www.scipy.org/stackspec.html.

We recommend installing with conda so that dependency installation is not an issue:

```
$ conda install -c moorepants dynamicisttoolkit
```

You can install using pip. Pip will theoretically[1] get the dependencies for you (or at least check if you have them):

```
$ pip install DynamicistToolKit
```

Or download the source with your preferred method and install manually.

Using Git:

```
$ git clone git@github.com:moorepants/DynamicistToolKit.git
$ cd DynamicistToolKit
```

Or wget:

```
$ wget https://github.com/moorepants/DynamicistToolKit/archive/master.zip
$ unzip master.zip
$ cd DynamicistToolKit-master
```

Then for basic installation:

```
$ python setup.py install
```

Or install for development purposes:

```
$ python setup.py develop
```

[1] You will need all build dependencies and also note that matplotlib doesn't play nice with pip.

CHAPTER 6

Tests

Run the tests with nose:

```
$ nosetests
```

CHAPTER 7

Vagrant

A vagrant file and provisioning script are included to test the code on an Ubuntu 13.10 box. To load the box and run the tests simply type:

```
$ vagrant up
```

See `bootstrap.sh` and `VagrantFile` to see what's going on.

CHAPTER 8

Documentation

The documentation is hosted at ReadTheDocs:

http://dynamicisttoolkit.readthedocs.org

You can build the documentation (currently sparse) if you have Sphinx and numpydoc:

```
$ cd docs
$ make html
$ firefox _build/html/index.html
```

CHAPTER 9

Release Notes

0.5.2

- Screwed up pypi upload on 0.5.1, so bumping one more time.

0.5.1

- Import nanmean from numpy instead of scipy and fix float slices. [PR #34]

0.5.0

- bicycle.py functions now output numpy arrays instead of matrices.
- Support for Python 3 [PR #30 and #32].

0.4.0

- Made the numerical derivative function more robust and featureful. [PR #27]
- `butterworth` now uses a corrected cutoff frequency to adjust for the double filtering. [PR #28]

0.3.5

- Fixed bug in coefficient_of_determination. [PR #23]

0.3.4

- Fixed bug in normalized cutoff frequency calculation. [PR #21]

0.3.2

- Fixed bug in butterworth function and added tests.

0.3.1

- Fixed butterworth to work with SciPy 0.9.0. [PR #18]

0.3.0

- Removed pandas dependency.
- Improved time vector function.
- Removed gait analysis code (walk.py), now at http://github.com/csu-hmc/Gait-Analysis-Toolkit.
- TravisCI tests now run, added image to readme.
- Added documentation at ReadTheDocs.

0.2.0

- Addition of walking dynamics module.

0.1.0

- Original code base that was used for the computations in this dissertation: https://github.com/moorepants/dissertation

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Bibliography

[Basu-Mandal2007] Basu-Mandal, P.; Chatterjee, A. & Papadopoulos, J. M. Hands-free circular motions of a benchmark bicycle. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 2007, 463, 1983-2003

[Meijaard2007] Meijaard, J. P.; Papadopoulos, J. M.; Ruina, A. & Schwab, A. L. Linearized dynamics equations for the balance and steer of a bicycle: A benchmark and review. Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences, 2007, 463, 1955-1982

[Moore2012] Moore, J. K. Human Control of a Bicycle. PhD Dissertation. University of California, Davis, 2012

Python Module Index

d

Index