API Documentation

API Documentation

June 27, 2011

Contents

Contents **1**

1 Package yeadon **2**
1.1 Modules . . . 2

2 Module yeadon.densities **3**
2.1 Variables . . . 3

3 Module yeadon.human **4**
3.1 Class human . . . 4
3.1.1 Methods . . . 4

4 Module yeadon.inertia **6**
4.1 Functions . . . 6

5 Module yeadon.measurements **9**
5.1 Variables . . . 9

6 Module yeadon.mymath **12**
6.1 Functions . . . 12
6.2 Variables . . . 13

7 Module yeadon.segment **14**
7.1 Class segment . . . 14
7.1.1 Methods . . . 14

8 Module yeadon.solid **16**
8.1 Class stadium . . . 16
8.1.1 Methods . . . 16
8.2 Class solid . . . 16
8.2.1 Methods . . . 17
8.3 Class stadiumsolid . . . 17
8.3.1 Methods . . . 18
8.4 Class semiellipsoid . . . 19
8.4.1 Methods . . . 19

9 Module yeadon.yeadon **21**
9.1 Functions . . . 21

9.2 Variables 21

1 Package yeadon

1.1 Modules

- **densities** *(Section 2, p. 3)*
- **human** *(Section 3, p. 4)*
- **inertia** *(Section 4, p. 6)*
- **measurements** *(Section 5, p. 9)*
- **mymath** *(Section 6, p. 12)*
- **segment** *(Section 7, p. 14)*
- **solid** *(Section 8, p. 16)*
- **yeadon** *(Section 9, p. 21)*

2 Module yeadon.densities

2.1 Variables

Name	Description
LitersPerCupicMeter	**Value:** `1000.0`
secondconversion	**Value:** `1.0`
DensityOfSegments	**Value:** `np.zeros(10)`
DensityOfSegmentsConverted	**Value:** `DensityOfSegments* LitersPerCupicMeter* secondconversion`
Ds	**Value:** `np.zeros(8)`
Da	**Value:** `np.zeros(7)`
Db	**Value:** `np.zeros(7)`
Dj	**Value:** `np.zeros(9)`
Dk	**Value:** `np.zeros(9)`

3 Module yeadon.human

3.1 Class human

3.1.1 Methods

__init__(*self*, *meas*, *DOF*)

```
Initializes a human object. Stores inputs as instance variables, defines the names of the degrees

Parameters
----------
meas : python module
    Holds roughly 95 measurements (supposedly in meters) that allow the generation of stadium soli
DOF : dictionary with 21 entries
    The degrees of freedom of the human (radians).
```

updateSegments(*self*)

Updates all segments. Expected to be called by a GUI or commandline interface when a user has updated joint angles after the human object has been created. Solids do not need to be recreated, but the segments need to be redefined, and so inertia parameters need to be averaged again, and the human's inertia parameters must also be redefined.

validateDOFs(*self*)

```
Validates the joint angle degrees of freedom against the DOF bounds specified in the definition of

Returns
-------
boolval : boolean
    0 if all DOFs are okay, -1 if there is an issue
```

averageSegmentProperties(*self*)

Yeadon 1989-ii mentions that the model is to have symmetric inertia properties, especially for the sake of modelling the particular aerial movement that is the subject of the Yeadon-i-iv papers. This function sets the mass and relative/local inertia tensor of each limb (arms and legs) to be the average of the left and right limbs.

calcProperties(*self*)

Calculates the mass, center of mass, and inertia tensor of the human. The quantities are calculated from the segment quantities. This method also calculates quantities in terms of the standard bicycle coordinate frame (x forward, z down).

printProperties(*self*)

Prints human mass, center of mass,and inertia.

draw(*self*)

Draws a self by calling the draw methods of all of the segments. Drawing is done by the matplotlib library.

drawOctant(*self*, *ax*, *u*, *v*, *c*)

```
Draws an octant of sphere. Assists with drawing the center of mass sphere.

Parameters
----------
ax : Axes3D object
    Axes on which to plot, defined by human.plot(self).
u : numpy.array
v : numpy.array
c : string
    Color
```

defineTorsoSolids(*self*)

Defines the solids (from solid.py) that create the torso of the human. This requires the definition of 2-D stadium levels using the input measurement parameters (meas.).

defineArmSolids(*self*)

Defines the solids (from solid.py) that create the arms of the human. This requires the definition of 2-D stadium levels using the input measurement parameters (meas.).

defineLegSolids(*self*)

Defines the solids (from solid.py) that create the legs of the human. This requires the definition of 2-D stadium levels using the input measurement parameters (meas.).

defineSegments(*self*)

Define segment objects using previously defined solids. This is where the definition of segment position and rotation really happens. There are 9 segments. Each segment has a base, located at a joint, and an orientation given by the input joint angle parameters.

writeMeasForISEG(*self*, *fname*)

Converts measurement input from the current format to the format used by Yeadon's Fortran code ISEG01B.F

4 Module yeadon.inertia

4.1 Functions

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 about of the body about a point located by d.
```

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 orthoganl
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
```

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.
```

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.
```

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.
```

rotate_inertia_tensor(*I*, *angle*)

Returns inertia tensor rotated through angle. Only for 2D

5 Module yeadon.measurements

5.1 Variables

Name	Description
Ls0p	**Value:** 5.3
Ls0w	**Value:** 2.0
s0h	**Value:** 2.0
Ls1p	**Value:** 4.8
Ls1w	**Value:** 1.8
s1h	**Value:** 0.8
Ls2p	**Value:** 5.6
Ls2w	**Value:** 2.0
s2h	**Value:** 3.0
Ls3p	**Value:** 6.3
Ls3w	**Value:** 2.3
s3h	**Value:** 3.0
Ls4d	**Value:** 6.8
Ls4w	**Value:** 2.5
s4h	**Value:** 0.6
Ls5p	**Value:** 5.7
Ls5w	**Value:** 2.0
s5h	**Value:** 1.0
Ls6p	**Value:** 3.0
s6h	**Value:** 1.
Ls7p	**Value:** 5.0
s7h	**Value:** 1.6
La0p	**Value:** 3.0
a0h	**Value:** 2.0
La1p	**Value:** 3.0
a1h	**Value:** 1.5
La2p	**Value:** 3.0
a2h	**Value:** 1.0
La3p	**Value:** 3.0
a3h	**Value:** 2.0
La4p	**Value:** 3.0
La4w	**Value:** 1.0
a4h	**Value:** 0.5
La5p	**Value:** 3.0
La5w	**Value:** 1.0
a5h	**Value:** 0.6
La6p	**Value:** 3.0
La6w	**Value:** 1.0
a6h	**Value:** 0.6
La7p	**Value:** 3.0
La7w	**Value:** 1.0
Lb0p	**Value:** 3.0
b0h	**Value:** 2.0
Lb1p	**Value:** 3.0
b1h	**Value:** 1.5
Lb2p	**Value:** 3.0

continued on next page

Name	Description
b2h	**Value:** `1.0`
Lb3p	**Value:** `3.0`
b3h	**Value:** `2.0`
Lb4p	**Value:** `3.0`
Lb4w	**Value:** `1.0`
b4h	**Value:** `0.5`
Lb5p	**Value:** `3.0`
Lb5w	**Value:** `1.0`
b5h	**Value:** `0.6`
Lb6p	**Value:** `3.0`
Lb6w	**Value:** `1.0`
b6h	**Value:** `0.6`
Lb7p	**Value:** `3.0`
Lb7w	**Value:** `1.0`
Lj0p	**Value:** `3.0`
j0h	**Value:** `1.0`
Lj1p	**Value:** `2.8`
j1h	**Value:** `2.0`
Lj2p	**Value:** `2.6`
j2h	**Value:** `2.0`
Lj3p	**Value:** `2.4`
j3h	**Value:** `1.2`
Lj4p	**Value:** `2.2`
j4h	**Value:** `2.7`
Lj5p	**Value:** `2.0`
j5h	**Value:** `0.4`
Lj6p	**Value:** `2`
Lj6w	**Value:** `.8`
j6h	**Value:** `0.4`
Lj7p	**Value:** `2.0`
j7h	**Value:** `0.5`
Lj8p	**Value:** `2`
Lj8w	**Value:** `.8`
j8h	**Value:** `0.5`
Lj9p	**Value:** `2`
Lj9w	**Value:** `.8`
Lk0p	**Value:** `3.0`
k0h	**Value:** `1.0`
Lk1p	**Value:** `2.8`
k1h	**Value:** `2.0`
Lk2p	**Value:** `2.6`
k2h	**Value:** `2.0`
Lk3p	**Value:** `2.4`
k3h	**Value:** `1.2`
Lk4p	**Value:** `2.2`
k4h	**Value:** `2.7`
Lk5p	**Value:** `2.0`
k5h	**Value:** `0.4`
Lk6p	**Value:** `2`
Lk6w	**Value:** `.8`

continued on next page

Name	Description
k6h	**Value:** `0.4`
Lk7p	**Value:** `2.0`
k7h	**Value:** `0.5`
Lk8p	**Value:** `2`
Lk8w	**Value:** `.8`
k8h	**Value:** `0.5`
Lk9p	**Value:** `2`
Lk9w	**Value:** `0.8`

6 Module yeadon.mymath

6.1 Functions

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 w

Returns
-------
R : numpy.matrix, shape(3,3)
        Three dimensional rotation matrix about three different orthogonal axes.
```

RotateRel(*angles*)

```
The three-dimensional relative rotation matrix from Yeadon 1989-i used to describe the orientation

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 w

Returns
-------
R : numpy.matrix, shape(3,3)
        Three dimensional rotation matrix about three different orthogonal axes.
```

RotateInertia(*RotMat*, *relInertia*)

```
Rotates an inertia tensor. A derivation of the formula in this function can be found in Crandall 1

Parameters
----------
RotMat : numpy.matrix, shape(3,3)
        Three-dimensional rotation matrix specifying the coordinate frame that the input inertia t
relInertia : numpy.matrix, shape(3,3)
        Three-dimensional cartesian inertia tensor describing the inertia of a mass in a rotated c

Returns
-------
Inertia : numpy.matrix, shape(3,3)
        Inertia tensor with respect to a fixed coordinate system ("unrotated").
```

6.2 Variables

Name	Description
zunit	**Value:** `np.array([[0], [0], [1]])`

7 Module yeadon.segment

7.1 Class segment

7.1.1 Methods

__init__(*self*, *label*, *pos*, *RotMat*, *solids*, *color*)

```
Initializes a segment object. Stores inputs as instance variables, calculates the orientation of t

Parameters
----------
label : str
    The ID and name of the segment.
pos : numpy.array, shape(3,1)
        The vector position of the segment's base, with respect to the fixed human frame.
Rotmat : numpy.matrix, shape(3,3)
        The orientation of the segment is given by a rotation matrix that specifies the orientatio
solids : list of solid objects
        The solid objects that compose the segment
color : str
        Color with which to plot this segment in the plotting functions.
```

setOrientations(*self*)

Sets the position (self.pos) and rotation matrix (self.RotMat) for all solids in the segment by calling each constituent solid's setOrientation method. The position of the i-th solid, expressed in the fixed human reference frame, is given by the sum of the segment's base position and the directed height of all the solids of the segment up to the i-th solid.

calcRelProperties(*self*)

Calculates the mass, relative/local center of mass, and relative/local inertia tensor (about the segment's center of mass). Also computes the center of mass of each constituent solid with respect to the segment's base in the segment's reference frame.

calcProperties(*self*)

Calculates the segment's center of mass with respect to the fixed human frame origin (in the fixed human reference frame) and the segment's inertia in the fixed human frame but about the segment's center of mass.

printProperties(*self*)

Prints mass, center of mass (in segment's and fixed human frames), and inertia (in segment's and fixed human frames).

printSolidProperties(*self*)

Calls the printProperties() member method of each of this segment's solids. See the solid class's definition of printProperties(self) for more detail.

draw(*self*, *ax*)

Draws all the solids within a segment.

8 Module yeadon.solid

8.1 Class stadium

Stadium

8.1.1 Methods

__init__(*self*, *inID*, *in1*, *in2*, *alignment*='ML')

```
Defines a 2D stadium shape and checks inputs for errors. A stadium, described in Yeadon 1989-ii, is

Parameters
----------
inID : str
    Identifies the type of information for the next two inputs. 'perimwidth' for perimeter and widt
in1 : float
    Either perimeter, depth, or thickness, as determined by inID
in2 : float
    Either width, or radius, as determined by inID
First argument is the perimeter p, and the second is the total width w. If an optional third argume
alignment = 'ML' : str
    Identifies the long direction of the stadium. 'ML' stands for medio-lateral. Aleternatively, '
```

plot(*self*, *ax*, *c*)

```
Plots the 2D stadium on 3D axes.

Parameters
----------
ax : Axes3D object
    Axis object from the matplotlib library.
c : str
    Color
```

8.2 Class solid

Solid

8.2.1 Methods

__init__(*self*, *label*, *density*)

```
Defines a solid. This is a base class. Sets the alpha value to be used for plotting with matplotli

Parameters
----------
label : str
    Name of the solid
density : float
    In units (kg/m^3), used to calculate the solid's mass
```

setOrientation(*self*, *pos*, *RotMat*)

Sets the position, rotation matrix of the solid, and calculate the "absolute" properties (center of mass, and inertia tensor) of the solid.

calcProperties(*self*)

Sets the center of mass and inertia of the solid, both with respect to the fixed human frame.

printProperties(*self*)

Prints the mass, center of mass (local and absolute), and inertia tensor (local and absolute) of the solid.

draw(*self*, *ax*, *c*)

8.3 Class stadiumsolid

yeadon.solid.solid
└ **yeadon.solid.stadiumsolid**

Stadium solid. Derived from the solid class.

8.3.1 Methods

__init__(*self*, *label*, *density*, *stadium0*, *stadium1*, *height*)

```
Defines a stadium solid object. Creates its base object, and calculates relative/local inertia pro

Parameters
----------
label : str
    Name of the solid.
density : float
    Density of the solid (kg/m^3).
stadium0 : stadium object
    Lower stadium of the stadium solid.
stadium1 : stadium object
    Upper stadium of the stadium solid.
height : float
    Distance between the lower and upper stadia.
```

Overrides: yeadon.solid.solid.__init__

calcRelProperties(*self*)

Calculates mass, relative center of mass, and relative/local inertia, according to formulae in Appendix B of Yeadon 1989-ii. If the stadium solid is arranged anterior-posteriorly, the inertia is rotated by pi/2 about the z axis.

draw(*self*, *ax*, *c*)

Draws stadium solid using matplotlib's mplot3d library. Plotted with a non-one value for alpha. Also places the solid's label near the center of mass of the solid. Adjusts the plot for solids oriented anterior-posteriorly. Plots coordinate axes of the solid at the base of the solid.

Overrides: yeadon.solid.solid.draw

makePos(*self*, *i*)

Generates coordinates to be used for matplotlib purposes.

F1(*self*, *a*, *b*)

Integration term. See Yeadon 1990-ii Appendix 2.

F2(*self*, *a*, *b*)

Integration term. See Yeadon 1990-ii Appendix 2.

F3(*self*, *a*, *b*)

Integration term. See Yeadon 1990-ii Appendix 2.

F4(*self*, *a*, *b*)

Integration term. See Yeadon 1990-ii Appendix 2.

F5(*self*, *a*, *b*)

Integration term. See Yeadon 1990-ii Appendix 2.

Inherited from yeadon.solid.solid(Section 8.2)

calcProperties(), printProperties(), setOrientation()

8.4 Class semiellipsoid

yeadon.solid.solid
└
yeadon.solid.semiellipsoid

Semiellipsoid.

8.4.1 Methods

__init__(*self*, *label*, *density*, *baseperim*, *height*)

```
Defines a semiellipsoid (solid) object. Creates its base object, and calculates rela

Parameters
----------
label : str
    Name of the solid.
density : float
    Density of the solid (kg/m^3).
baseperimeter : float
    The base is circular.
radius : float
    Calculated for base perimeter.
height : float
    The remaining minor axis.
```

Overrides: yeadon.solid.solid.__init__

calcRelProperties(*self*)

Calculates mass, relative center of mass, and relative/local inertia, according to formulae in Appendix B of Yeadon 1989-ii.

draw(*self*, *ax*, *c*)

Draws semiellipsoid using matplotlib's mplot3d library. Plotted with a non-one value for alpha. Also places the solid's label near the center of mass of the solid. Plots coordinate axes of the solid at the base of the solid. Code is modified from matplotlib documentation for mplot3d.

Overrides: yeadon.solid.solid.draw

Inherited from yeadon.solid.solid(Section 8.2)

calcProperties(), printProperties(), setOrientation()

9 Module yeadon.yeadon

9.1 Functions

modifyJointAngles()

Called by command-line interaction to modify joint angles. Allows the user to first select a joint angle (from the dictionary DOF) to modify. Then, the user inputs a new value for that joint angle in units of pi-radians. The user continues to modify joint angles until the user quits. The user can quit at any time by entering q.

printSegmentProperties()

Called by commandline interaction to choose a segment to print the properties (mass, center of mass, inertia), and to print those properties. See the documentation for the segment class for more information. The user can print properties of segments endlessly until entering q.

printSolidProperties()

Called by commandline interaction to print the properties (mass, center of mass, inertia) of a solid chosen by user inputs. The user first selects a segment, and then chooses a solid within that segment. Then, the properties of that solid are shown. See the documentation for the solid class for more information.

9.2 Variables

Name	Description
H	**Value:** `hum.human(meas, DOF)`
done	**Value:** `0`
frames	**Value:** `'Yeadon', 'bike'`
frame	**Value:** `0`
nonfr	**Value:** `1`
DOF	**Value:** `{'somersalt': np.pi/ 2* 0.2, 'tilt': 0.0, 'twist': 0.0, '...`

Index

yeadon *(package)*, 2
- yeadon.densities *(module)*, 3
- yeadon.human *(module)*, 4–5
  - yeadon.human.human *(class)*, 4–5
- yeadon.inertia *(module)*, 6–8
  - yeadon.inertia.cylinder_inertia *(function)*, 7
  - yeadon.inertia.inertia_components *(function)*, 6
  - yeadon.inertia.parallel_axis *(function)*, 6
  - yeadon.inertia.rotate_inertia_tensor *(function)*, 8
  - yeadon.inertia.total_com *(function)*, 7
  - yeadon.inertia.tube_inertia *(function)*, 6
- yeadon.measurements *(module)*, 9–11
- yeadon.mymath *(module)*, 12–13
  - yeadon.mymath.Rotate3 *(function)*, 12
  - yeadon.mymath.RotateInertia *(function)*, 12
  - yeadon.mymath.RotateRel *(function)*, 12
- yeadon.segment *(module)*, 14–15
  - yeadon.segment.segment *(class)*, 14–15
- yeadon.solid *(module)*, 16–20
  - yeadon.solid.semiellipsoid *(class)*, 19–20
  - yeadon.solid.solid *(class)*, 16–17
  - yeadon.solid.stadium *(class)*, 16
  - yeadon.solid.stadiumsolid *(class)*, 17–19
- yeadon.yeadon *(module)*, 21
  - yeadon.yeadon.modifyJointAngles *(function)*, 21
  - yeadon.yeadon.printSegmentProperties *(function)*, 21
  - yeadon.yeadon.printSolidProperties *(function)*, 21