yeadon Documentation

Release 1.4.0

Chris Dembia

Oct 18, 2022

Contents

This package calculates the masses, center of mass positions, and inertia tensors that correspond to the human inertia model developed by Yeadon [1]. The package allows for the input of both measurements from human subjects and configuration variables (joint angles) with which one can orient the model. Additionally, the package allows for 3D visualization of the model using the package.

One possible use of the package is to incorporate the inertial properties of an actual human into a rigid body dynamics model that contains a human. Then, the model containing the human can be compared to experiments performed with the same human.

This package was developed during the Summer of 2011 at the University of California, Davis, to aid with the bicycle research of Jason Moore and Dale Luke Peterson in the Sports Biomechanics Lab of Professor Mont Hubbard. Jason Moore had a multibody dynamics model of a human riding a bicycle, and performed experiments with humans riding a bicycle. To compare his model to his experiments, he needed the inertial properties of the human riding the bicycle. That's what this package was able to provide him [2]. Learn more about the Sports Biomechanics Lab at biosport.ucdavis.edu.

Here is a video that introduces the basics of this package: http://youtu.be/o-5Ss6YLY0I.

CHAPTER 1

Contents

1.1 Overview

This page describes the basics of Yeadon's inertia model. It is expected that the user of this package has read Yeadon's 1990 papers, especially Yeadon 1990-ii. There are four related papers, identified by numerals i-iv.

Here is a summary of his four papers:

- i: motivation, conceptual description of joints, obtaining orientation angles from film
- ii: modeling human geometry using stadium solids
- iii: inertia transforms and angular momentum of the stadium solids
- iv: simulation verification

The *Measurements* page in this documentation describes the particularly relevant parts of paper ii, while the *Configuration* page does the same for paper iv.

Yeadon models a human using 39 stadium solids and 1 semi-ellipsoid (for the head). These 40 solids make up 11 rigid body segments, which are connected to each other via joints (e.g., the knee). This relatively simple geometry allows for one to swiftly calculate quantities relevant for dynamics. These quantities are mass, center of mass positions, and inertia tensors. These quantities can be obtained in the global reference frame, or in the local frame of a segment or solid.

One can use this package to incorporate a human into equations of motion, though this endeavor is left to the user. The package does not deal at all with angular momentum (the topic of paper iii).

There are a few differences between Yeadon's model described in his publications and the model implemented in this package. Here are some of the bigger ones:

- In Yeadon's model, the global frame of the human is defined in a complicated way that depends on the configuration of the human. In this package, the global frame does not depend on the configuration.
- In Yeadon's model, the orientation of the legs are related to each other, so that there are less degrees of freedom than there are joint angles (generalized coordinates). No joint angles are coupled in this package.

- Yeadon provides the option of formulating the model with additional joints for the feet and hands. Here, the feet and hands are rigid parts of the legs and forearms, respectively.
- Yeadon labels the *solids* with indices starting from 1 (s1 is the first solid), while this package indexes the solids from 0 (s0 is the first solid). The labels for the *segments* (e.g. A1, J1, etc.) are unchanged.
- Both packages allow making the model symmetric (averaging both the two arms and the two legs), but do so in different ways. We average the input measurements for the limbs, and then proceed to compute masses, center of mass positions, and inertia tensors with these averaged measurements. Yeadon, however, enforces symmetry by averaging these three quantities as the last step (the measurements across limbs are not averaged).

1.2 Usage

This page shows how one can use `yeadon` once it's installed.

1.2.1 Three different interfaces

There are three ways of using the `yeadon` package: through the text-based user interface (UI), through the graphical user interafce (GUI), and through a Python interpreter or in your own Python code. The can run the UI by entering the following in a terminal or command window:

```
$ yeadon --ui
```

or by entering a Python interpreter and executing the following:

```
import yeadon
yeadon.start_ui()
```

The interface will guide you through its use. You can enter in measurements, then configuration (joint angles), and then can modify joint angles, access data, or use one of the features listed below.

The GUI is run by entering the following in a terminal or command window:

```
$ yeadon --gui
```

or by entering a Python interpreter and executing the following:

```
>>> yeadon.start_gui()
```

The last way is through the API in a Python script or module. You import the module and then create a `Human`:

```
>>> import yeadon
>>> human = yeadon.Human(<measfilename>, <CFGfilename>)
```

where `<measfilename>` and `<CFGfilename>` are either paths to .txt files, or are dictionaries. The `<CFGfilename>` argument is optional. If not provided, the human is created in a default configuration. See *Measurements* or *Configuration* for more detail. With an instance of `Human`, we can access inertia properties of the entire human, of its segments (e.g. limbs), or of the individual solids that make up the segments.

1.2.2 Attributes of `Human`

Suppose we have an instance of `Human`, named `chad`. Before we show what one can do with a `Human`, we present the attributes that represent the human's segments. There is an attribute for each segment, whose name is the same as that used by Yeadon.

- `chad.P` pelvis
- `chad.T` thorax
- `chad.C` chest-head
- `chad.A1` left upper arm
- `chad.A2` left forearm-hand
- `chad.B1` right upper arm
- `chad.B2` right forearm-hand
- `chad.J1` left thigh
- `chad.J2` left shank-foot
- `chad.K1` right thigh
- `chad.K2` right shank-foot

Also, one can access a list of all these segments, perhaps for iteration, via the `chad.segments` attribute. The solids that make up each segment can be accessed through the `solids` attribute of each of the `Segments`'s above:

```
>>> chad.P.solids[0].label
's0: hip joint centre'
```

1.2.3 Setting the configuration

One can set the configuration of the model using a `<CFGfilename>` as described above, or by using the `chad.set_CFG()` method:

```
>>> chad.set_CFG('somersault', 0.5 * 3.1416)
```

When one calls this method, the inertia properties are recomputed. The list of configuration variables is stored in `Human.CFGnames`.

1.2.4 Summary of functionality

Print inertia properties

This is the quickest way to get the relevant information out of the model. There are methods to print the properties of the entire human, of segments, or of solids. The following prints the inertia properties for the entire human:

```
>>> chad.print_properties()
Mass (kg): 58.2004885884

COM in global frame from bottom center of pelvis (Ls0) (m):
[[  1.62144613e-17]
[  0.00000000e+00]
[  1.19967938e-02]]

Inertia tensor in global frame about human's COM (kg-m^2):
[[  9.63093850e+00   2.20795600e-20   6.10622664e-16]
[  2.20795600e-20   9.99497872e+00   2.70396625e-36]
[  6.10622664e-16   2.70396625e-36   5.45117742e-01]]
```

The following shows how one can print the inertia properties for the `J1`, or left thigh, segment:

```
>>> chad.J1.print_properties()
J1: Left thigh properties:

Mass (kg): 8.50477532204

COM in segment's frame from segment's origin (m):
[[ 0.        ]
[ 0.        ]
[ 0.19276748]]

COM in global frame from bottom center of pelvis (Ls0) (m):
[[ 0.081     ]
[ 0.        ]
[-0.19276748]]

Inertia tensor in segment's frame about segment's COM (kg-m^2):
[[ 0.14109999  0.          0.        ]
[ 0.          0.14109999  0.        ]
[ 0.          0.          0.02718329]]

Inertia tensor in global frame about segment's COM (kg-m^2):
[[  1.41099994e-01   0.00000000e+00   1.39507727e-17]
[  0.00000000e+00   1.41099994e-01   0.00000000e+00]
[  1.39507727e-17   0.00000000e+00   2.71832899e-02]]
```

Lastly, there is a method for each segment that prints the inertia properties of the individual solids that make up the segment (output not shown):

```
>>> chad.J1.print_solid_properties()
```

Below, we delve into more detail about what these quantities are.

Return inertia properties

It may be desirable to directly access the kinematics information and inertia properties from the attributes. Below, we show the docstrings for these properties, as can be accessed in an IPython interpreter. Also, one can obtain iinformation about the data type of the properties using `help(<property>)` (e.g., `help(chad.mass)`). The docstrings make reference to *the bottom center of the pelvis (Ls0)*, the *origin of the segment/solid*; and the *global* and *segment* frames. These locations and frames are descrbed in *Configuration*.

There are three inertia properties for the human overall:

```
>>> chad.mass?
...Docstring:  Mass of the human, in units of kg....

>>> chad.center_of_mass?
...Docstring: Center of mass of the human, a np.ndarray, in units of m,
expressed the global frame, from the bottom center of the pelvis
(center of the Ls0 stadium)....

>>> chad.inertia?
...Docstring: Inertia matrix/dyadic of the human, a np.matrix, in units
of kg-m^2, about the center of mass of the human, expressed in the
global frame....
```

For each segment, there are five properties that are related to inertia, and three related strictly to kinematics:

```
>>> chad.J1.mass?
...Docstring:  Mass of the segment, in units of kg....

>>> chad.J1.rel_center_of_mass?
...Docstring: Center of mass of the segment, a np.ndarray, in units of
m, expressed in the frame of the segment, from the origin of the
segment....

>>> chad.J1.center_of_mass?
...Docstring: Center of mass of the segment, a np.ndarray, in units of
m, expressed in the global frame, from the bottom center of the
pelvis....

>>> chad.J1.rel_inertia?
...Docstring: Inertia matrix/dyadic of the segment, a np.matrix, in
units of kg-m^2, about the center of mass of the segment, expressed in
the frame of the segment....

>>> chad.J1.inertia?
...Docstring: Inertia matrix/dyadic of the segment, a np.matrix, in
units of kg-m^2, about the center of mass of the human, expressed in
the global frame....

>>> chad.J1.pos?
...Docstring: Position of the origin of the segment, a np.ndarray, in
units of m, expressed in the global frame, from the bottom center of
the pelvis (Ls0)....

>>> chad.J1.end_pos?
...Docstring: Position of the center of the last (farthest from pelvis)
stadium in this segment, a np.ndarray, in units of m, expressed in the
global frame, from the bottom center of the pelvis (Ls0)....

>>> chad.J1.rot_mat?
...Docstring: Rotation matrix specifying the orientation of this
segment relative to the orientation of the global frame, a np.matrix,
unitless.  Multiplying a vector expressed in this segment's frame with
this rotation matrix on the left gives that same vector, but expressed
in the global frame....
```

The attributes for the solids are similar to those for the segments, except that they do not have a `rot_mat` attribute (their `rot_mat` is that of the segment containing them):

```
>>> chad.J1.solids[0].mass?
...Docstring: Mass of the solid, in units of kg....

>>> chad.J1.solids[0].center_of_mass?
...Docstring: Center of mass of the solid, a np.ndarray, in units of m,
expressed in the global frame, from the bottom center of the pelvis
(Ls0)....

>>> chad.J1.solids[0].inertia?
...Docstring: Inertia matrix/dyadic of the solid, a np.matrix, in units
of kg-m^2, about the center of mass of the human, expressed in the
global frame....

>>> chad.J1.solids[0].rel_center_of_mass?
```

(continues on next page)

(continued from previous page)

```
...Docstring: Center of mass of the solid, a np.ndarray, in units of m,
expressed in the frame of the solid, from the origin of the solid....

>>> chad.J1.solids[0].rel_inertia?
...Docstring: Inertia matrix/dyadic of the solid, a np.matrix, in units
of kg-m^2, about the center of mass of the solid, expressed in the
frame of the solid....

>>> chad.J1.solids[0].pos?
...Docstring: Position of the origin of the solid, which is the center
of the surface closest to the pelvis, a np.ndarray, in units of m,
expressed in the global frame, from the bottom center of the pelvis
(Ls0)....

>>> chad.J1.solids[0].end_pos?
...Docstring: Position of the point on the solid farthest from the
origin along the longitudinal axis of the segment, a np.ndarray, in
units of m, expressed in the global frame, from the bottom center of
the pelvis (Ls0)....
```

Draw

One can create a window with a 3D rendering of the human model. The rendering portrays the human with the given measurements and specified configuration:

```
>>> chad.draw()
```

Scale by mass

The mass of the human that we calculate probably doesn't match that of the actual human subject being modeled. We calculate this mass using densities from literature. If you measure the human's actual mass and want to use that in *yeadon*, we can change the model's mass to this measured mass by scaling these densities. This can be done via the measurement input file by providing a positive value for `totalmass` (see measurement file template) or by a call to the `chad.scale_human_by_mass()` method.

Symmetry

One can average the measurements for the left and right limbs to create symmetrical limbs. This may be desirable depending on your use of the package. This symmetry is imposed by default. It can be changed by setting the keyword argument `symmetric` of the `Human` constructor to `False`. The symmetry of the model cannot be modified after the `Human` is constructed.

Combine inertia

One can obtain inertia properties for a combination of solids and/or segments. This is done via the `chad.combine_inertia()` method. See *API Documentation* for more information.

Transform inertia tensor

By default, the inertia tensor of the human is expressed in the global frame, whose origin is located at the bottom center of the pelvis (Ls0), and whose orientation is shown in *Configuration*, `draw()` and the GUI. To transform the inertia tensor so it's expressed in a different frame, you can use `chad.inertia_transformed()`.

File input/output

The measurements can be written to a text file using `chad.write_measurements()`. The configuration can be written using `chad.write_CFG()`. The measurements can be written to a text file that is ready for Yeadon's ISEG Fortran code using `chad.write_meas_for_ISEG()`.

1.3 Measurements

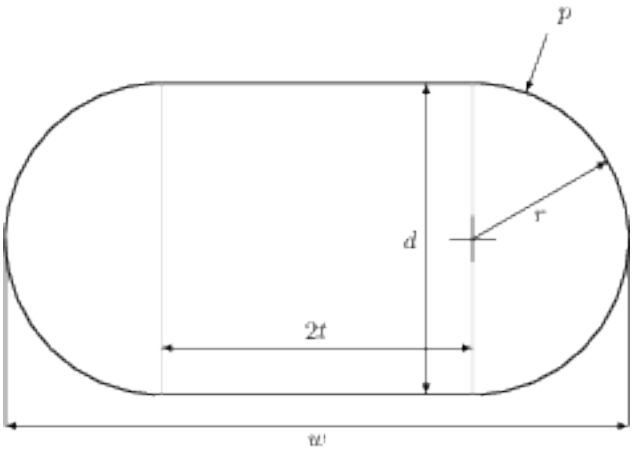
Here we describe the measurements that need to be taken to define a human with this model, and we provide some guidance for taking those measurements and getting them into the code.

1.3.1 The stadium shape and the stadium solid

The human is composed of 11 rigid body segments. Each segment is defined as a loft across a number of 2D parallel stadium shapes, which are defined below. In one case, a segment contains a semi-ellipsoid solid. The model is customized to an individual via 95 anthropomorphic measurements to define the stadia and the distances between them.

A *stadium* shape, show in the figure below, can be defined via any of the following 4 sets of 2 parameters:

- a radius r and thickness t,
- a perimeter p and width w along the stadium's longitudinal axis
- a perimeter p and a depth $d = 2r$.
- a depth d and a width w.



A circle can be defined by a stadium whose thickness is zero, $t = 0$.

Stadium solids are defined by two parallel stadia, as well as the height h of the solid between the two stadia (i.e., a loft between the two stadium cross sections).

1.3.2 Specification of all measurements

The figure below specifies all 95 measurements.

To define the stadium solids that make up the human model, one takes the measurements outlined here. The measurements consist of *lengths* L (different from heights), perimeters p, widths w, and depths d.

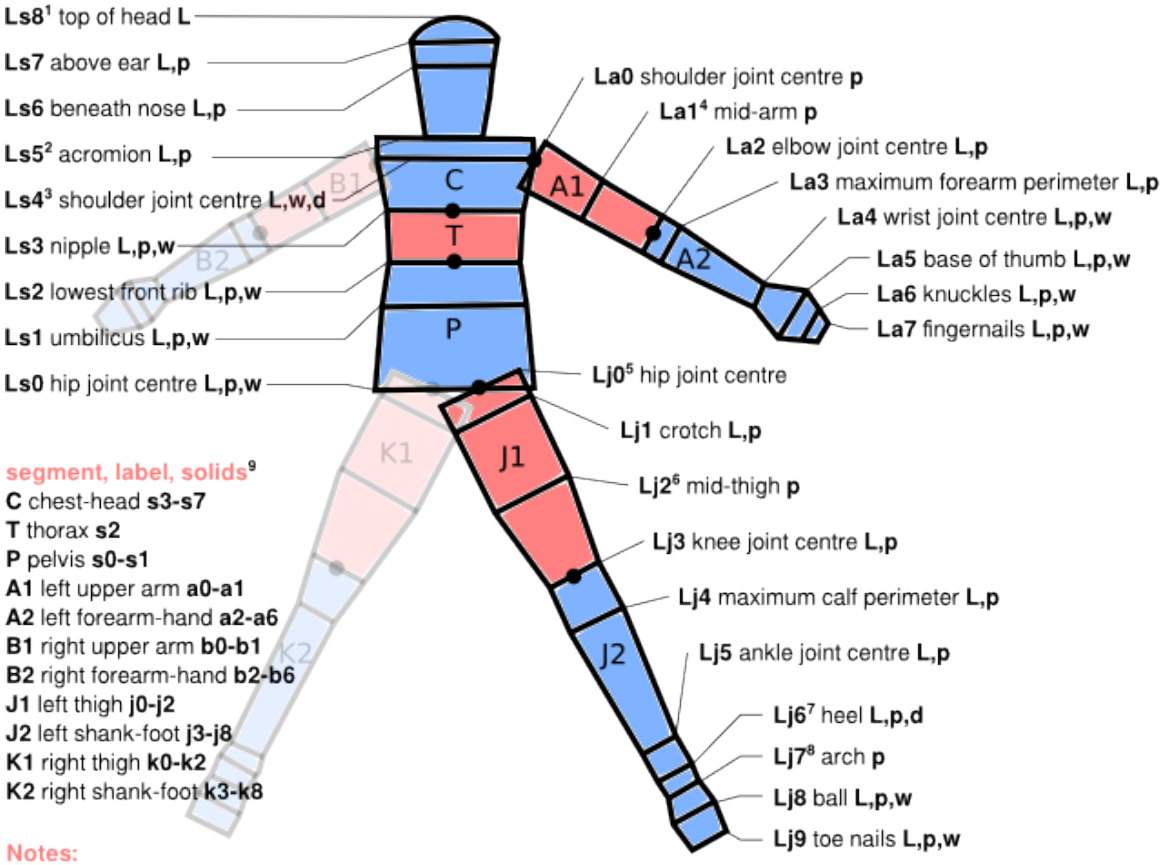
By measuring the parameters that define the stadia (also called *levels*), and the distance between these stadia, we define 39 stadium solids. Each stadium is shared by two stadium solids, except for the stadia at the end of the hands and feet. In general, the stadia are defined by measuring perimeter and width, since these are easier to measure. There are a few exceptions though, and these are described further down the page.

Key:

● denotes a joint centre
Segments P, T, C, A1, A2, B1, B2, J1, J2, K1, K2 are separated by alternating colors.
Levels are denoted as **L<s>#** with **<s>** roughly denoting a body part, and **#** denoting the **#**-th level in the segment.
Measurements are denoted as **L<s>#<t>**, with **<t>** denoting the type of measurement.

There are 4 types of measurements:
L denotes a length measurement, not necessarily measured from previous level:
Ls1L-Ls5L measured from **Ls0; Ls6L-Ls8L** measured from **Ls5**
La2L-La4L measured from **La0; La5L-La7L** measured from **La4** (same for segment **b**)
Lj1L, Lj3L-Lj5L measured from **Lj0; Lj6L, Lj8L-Lj9L** measured from **Lj5** (same for segment **k**)
p denotes a perimeter measurement, must have $2 < \mathbf{p}/\mathbf{w} < \pi$
w denotes a width (medio-lateral, or side to side) measurement
d denotes a depth (anterior-posterior, or front to back) measurement

level, label, measurements needed



segment, label, solids[9]
C chest-head **s3-s7**
T thorax **s2**
P pelvis **s0-s1**
A1 left upper arm **a0-a1**
A2 left forearm-hand **a2-a6**
B1 right upper arm **b0-b1**
B2 right forearm-hand **b2-b6**
J1 left thigh **j0-j2**
J2 left shank-foot **j3-j8**
K1 right thigh **k0-k2**
K2 right shank-foot **k3-k8**

Notes:
Total mass can be measured in order to scale the default densities used for the solids (otherwise, mass is estimated).
1 s0 is the only semi-ellipsoidal solid.
2 two stadia at this level, one for s4 and one for s5. The parameters for the s4 stadium are calculated from Ls4's stadium. Ls5 perimeter measured around neck.
3 depth is measured in lieu of perimeter since arms interfere and width is measured with respect to the shoulder joint centers.
4 La1L is not measured, but is instead set as half of La2L.
5 stadium (circle) perimeter is calculated from the dimensions of the stadium at Ls0.
6 Lj2L is not measured, but is instead set as the average of Lj1L and Lj3L.
7 Lj6's (and Lk6's) stadia are the only stadia oriented anteroposteriorly.
8 Lj7L is not measured, but is instead set as the average of Lj6L and Lj8L.
9 Yeadon's 1990 paper indexes the solids from 1, while this formulations indexes from 0.

It is lengths, not heights, that you measure off the subjects. That is, the length measurements are sums of heights, not the individual heights of the stadium solids. For example, The "length" for the Ls5 acromion level is measured

from Ls0, the hip joint centre, not from Ls4. The figure above lists the level from which the length for other levels are measured.

1.3.3 Scaling densities via a measured mass

The mass of the model is estimated from the measurements described above, along with densities for the various segments taken from the literature. In the case that you also measure the mass of the individual being modeled, it is possible to scale the densities so that the total mass of the human is that which you have measured. See *Usage* for a brief explanation on how to do this.

1.3.4 Exceptions to the general measurement practice

There are a number of exceptions to the general scheme of measurements required by the model.

- **Length exceptions**: Lengths to arm level 1 and leg levels 2 and 7 are not measured. The length from La0 to La1 (or Lb0 to Lb2) is set internally as half the length from La0 to La2 (or Lb0 to Lb2). The lengths to leg levels 2 and 7 are calculated as averages of the two lengths around leg levels 2 and 7. Thus, perimeters, etc. at these levels should be measured halfway between the surrounding levels (i.e., perimeter of the La1 stadium is measured at the point in in the arm halfway between La0 and La2).
- **Levels that are circles** (zero-thickness stadia): Arm levels 0-3 (the first four arm levels) and leg levels 0-5 and 7 (the first six and the arch). For these, only a perimeter measurement is required (no width or depth is measured).
- **Depth measurements**: As far as measurements are concerned, the only difference between a depth and a width is that a depth is measured anterior to posterior (front to back), while widths are measured medio-laterally (side to side) when the subject is in the configuration as drawn in the diagram above. Depths are measured at the Ls5 acromion, and the Lj6, Lk6 heel.
- **The neck**: The base of the neck, which is also located at level Ls5, acromion, is modeled as circular. Its radius is set internally from the acromion perimeter measurement. This means that the acromion perimeter should be measured about the base of the neck.

1.3.5 Getting measurements into the model

There are two options for getting measurements into the model:

- Use the *meastemplate.txt* input text file in the misc/ directory, or `here`, to define all measurements. The file uses the YAML syntax. This syntax allows you to treat the input file as a Python script in which you simply define a number of variables. See comments within the file for further details.
- Provide a python dictionary, containing all the appropriate fields, to the *yeadon.human.Human* constructor. You can obtain a sample dictionary from the variable `yeadon.human.Human.meas`. The keys for the dictionary are the names of the variables in the *meastemplate.txt* file, as strings.

Internally, the package uses units of meters for the measurements. However, it may be that you have worked with different units in gathering the measurements. In the case that you are using an input text file to specify measurements (i.e., *meastemplate.txt*), you can define the measurements using the units you desire, and the package will perform the unit conversion for you. This is done by providing a value for the variable `measurementconversionfactor` in the text file, as shown in *meastemplate.txt*. This is a number that converts the units of your measurements into meters. For example, if you took measurements in millimeters, you should give this variable the value 0.001. If you are providing measurements via a dictionary, the measurements must be in units of meters.

1.3.6 Sample measurement files

Here are measurement data files for three people we measured:

- `male1`
- `male2`
- `male3`
- `female1`

1.4 Configuration

The configuration of the human is set by 21 joint angles. The image below describes all the joints and the joint angles between the segments of the human.

There are two ways to provide the configuration to the package:

- A configuration text file, such as *CFGtemplate.txt* in misc/, or `here`. This file follows YAML syntax.
- A dictionary with the correct keys to the constructor of *`yeadon.human.Human`*. The keys are exactly as written in the image below. One can also access the `yeadon.human.Human.CFG` variable to see what the dictionary looks like.

1.4.1 Frames

Here, we use the term *frame* to mean a coordinate system: something with an origin and orientation (sometimes a frame does not include an origin, and is just a vector basis).

The **global frame** is located at the bottom center of the pelvis stadium, `Ls0`. The x and z axes are in a frontal plane, with the x axis directed to the left and the z axis oriented superiorly. The y axis is directed in the posterior direction.

Each segment has its own frame. The origins of the segment frames are denoted by the black dots, and are at the location of a joint center between two segments. Solids have their own frame as well, which share the same orientation of the segment containing them, but are shifted along the longitudinal (z) axis of the segment.

Each segment is rotated relative to its parent segment through body fixed x-y-z rotations (x-y-z Euler angles) as specified in M. R. Yeadon, "The simulation of aerial movement–i. The determination of orientation angles from film data.," Journal of biomechanics, vol. 23, no. 1, pp. 59–66, Jan. 1990.

1.4.2 Location of joint centers

Here is a description of the points at which segments are connected to each other (that is, the location of joint centers):

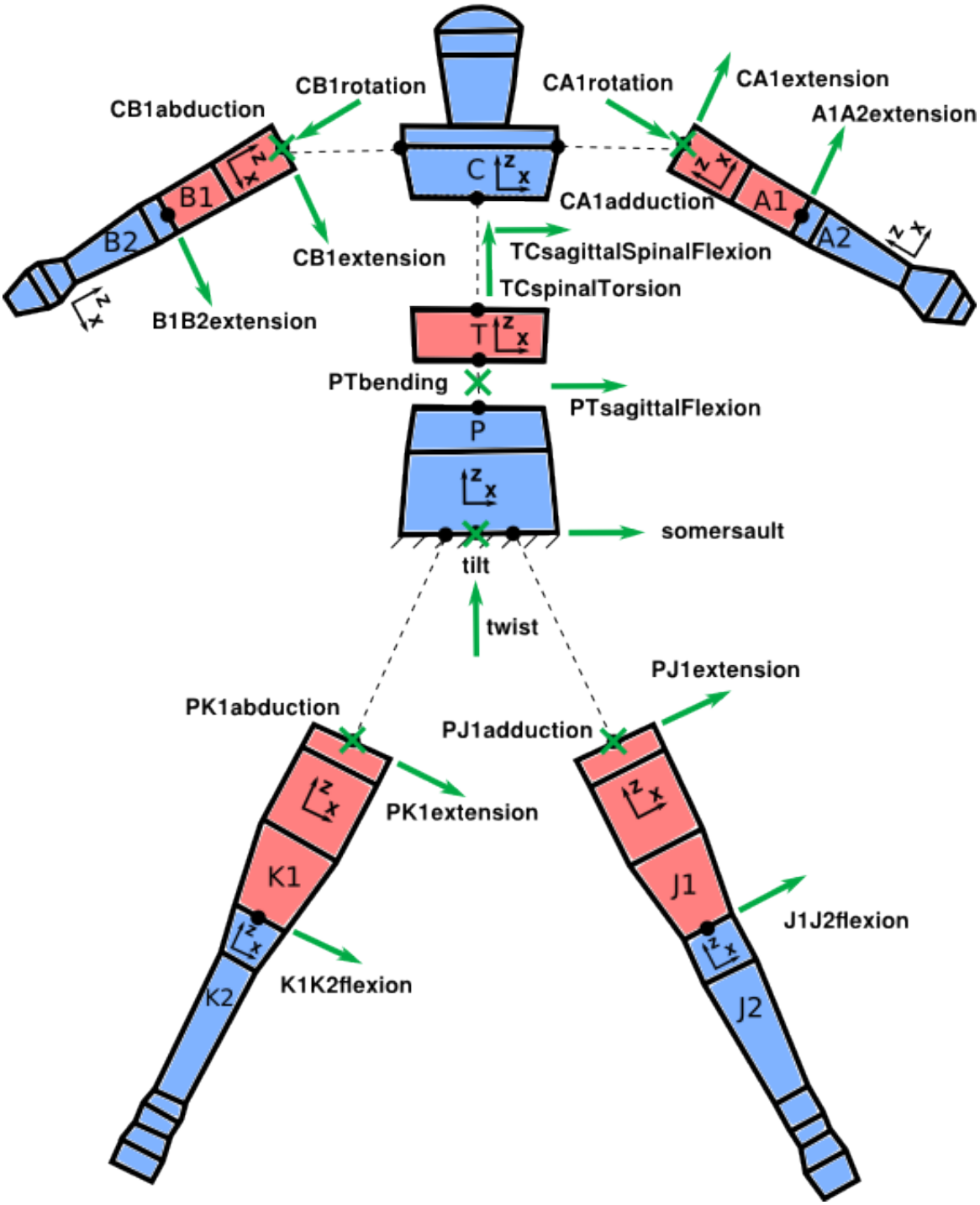
- The joint centers for the legs, for segments J1 and K1, are located at $(t, 0, 0)$ and $(-t, 0, 0)$ in the frame of the `s0` solid, respectively, where t is the thickness of the `Ls0` stadium (see *Measurements* for solid/stadium labels).
- The joint centers of the arms are at the Ls4 level, at the midpoint of the semicircular arcs of the Ls4 stadium.
- The two joint centers in the torso are centered along the z-axis of the stadia.

Notes/Key:

● denotes a joint centre

✕ is a rotation vector into the page

Black vectors denote the local coordinate frame of the segment they are within. Despite the vectors' locations, the local coordinate systems of the segments always have their origin at the segments' joint centre.
Each green vector represents a configuration variable. Use the right-hand-rule on these vectors to determine the positive direction of the configuration variable.



1.5 Release Notes

1.5.1 Future releases

See issues on github at https://github.com/chrisdembia/yeadon/issues.

1.5.2 v1.4.0

- Dropped support for Python < 3.7 (including 2.7).
- Replaced `yaml.load` with `yaml.safe_load`.
- Fixed pretty printing of results to work with newer NumPy versions.

1.5.3 v1.3.0

- Now supports Python 3.

1.5.4 v1.2.1

- Pinned the bicycle example to specific dependencies.
- Added version.py.
- Removed Mayavi print statements.
- Added badges to the README.
- Added citation note to the README.

1.5.5 v1.2

- Added two examples, PRs #98, #101.

1.5.6 v1.1

- Fixed somersault mispelling, issue #67.
- Now, configuration variables indicate proper sense. For example, flexion means flexion; not extension.
- Fixed serious bug in the computation of inertia tensors in different reference frames. The calculation of all solids' and segments' inertia tensors in the global frame was incorrect, PRs #79, #80.
- Now, use a consistent definition for rotation matrices: now, all rotation matrices `R` are of the form `v_a = R * v_b`, PR #88.
- Added a center of mass sphere to the GUI visualization, PR #95.
- Made mass center sphere and inertia ellipsoid off by default in the GUI, PR #93.
- Fixed default orientation of human in GUI visualization, PR #93.
- Improved the printing of human, segement, and solid properties, PR #81.
- Renamed `rotate3_inertia` to `rotate_inertia` and changed its definition to match the rotation matrix definitions in the rest of the software, PR #79.

- Setuptools now recommended, PR #72.
- `yeadon.__version__` now works, PR #69.
- Fixed Sphinx warnings in the docs.

1.5.7 v1.0

- Fairly thorough unit tests.
- Clarified documentation and docstrings.
- Improved the way rotation matrices are formed.
- Moved visualization to the MayaVi library, and introduced a GUI.
- Introduced the *yeadon.human.Human.inertia_transformed()* method.
- Use of python properties for inertia properties and other important attributes.
- Improved setup/build/installation process.

1.5.8 v0.8 on 18 July 2011

This is the first release.

1.6 API Documentation

The user only interacts with the *yeadon.human* module. The interface to the other modules is only useful to developers.

1.6.1 yeadon Package

human Module

The human module defines the Human class, which is composed of Segment's. The Human class has methods to define the constituent segments from inputs, calculates their properties, and manages file input/output.

class yeadon.human.**Human**(*meas_in*, *CFG=None*, *symmetric=True*, *density_set='Dempster'*)

Bases: object

CFGbounds = [[-3.141592653589793, 3.141592653589793], [-3.141592653589793, 3.141592653

CFGnames = ('somersault', 'tilt', 'twist', 'PTsagittalFlexion', 'PTbending', 'TCspinal'

calc_properties(*self*)

Calculates the mass, center of mass, and inertia tensor of the human. The quantities are calculated from the segment quantities.

center_of_mass

Center of mass of the human, a np.ndarray, in units of m, expressed the global frame, from the bottom center of the pelvis (center of the Ls0 stadium).

combine_inertia(*self*, *objlist*)

Returns the inertia properties of a combination of solids and/or segments of the human, using the fixed human frame (or the modified fixed frame as given by the user). Be careful with inputs: do not specify a solid that is part of a segment that you have also specified. This method does not assign anything to any object attributes (it is 'const'), it simply returns the desired quantities.

See documentation for description of the global frame.

Parameters

objlist [tuple] Tuple of strings that identify a solid or segment. The strings can be any of the following:

- solids: 's0' through 's7', 'a0' through 'a6', 'b0' through 'b6', 'j0' through 'j8', 'k0' through 'k8'
- segments: 'P', 'T', 'C', 'A1', 'A2', 'B1', 'B2', 'J1', 'J2', 'K1', 'K2'

Returns

combined_mass [float] Sum of the masses of the input solids and/or segments.

combined_COM [np.array (3,1)] Position of the center of mass of the input solids and/or segments, expressed in the global frame .

combined_inertia [np.matrix (3,3)] Inertia tensor about the combined_COM, expressed in the global frame.

draw(*self*, *mlabobj=None*, *gui=False*)

Draws the human in 3D in a new window using MayaVi. The mouse can be used to control or explore the 3D view.

Parameters

mlabobj [mayavi.mlab, optional, default=None] A mayavi mlab object. If None a new one will be created.

gui: boolean, optional, default=False If false the mlab.show() command will be called and the scene will be displayed to the screen.

get_segment_by_name(*self*, *name*)

Returns a segment given its name.

inertia

Inertia matrix/dyadic of the human, a np.matrix, in units of kg-m^2, about the center of mass of the human, expressed in the global frame.

inertia_transformed(*self*, *pos=None*, *rotmat=None*)

Returns an inertia tensor of the human with respect to the position provided in *pos* and a new frame that is defined by rotation relative to the global frame with the direction cosine matrix *rotmat*. The position is to be provided from the origin of the global frame, which is at the center of the Ls0 stadium (bottom of pelvis), and its components are expressed in the basis of the global frame. This method does NOT alter any attributes of the Human (it is 'const').

Parameters

pos [array_like, (3,), (1, 3), or (3, 1), optional] Position vector from the origin (center of Ls0) to the point about which the user desires the inertia tensor. This position vector must be expressed in the global reference frame. If not provided, the tensor is given about the center of mass of the human.

rotmat [np.matrix (3,3), optional] If not provided, the returned tensor is expressed in the global frame, else the returned tensor is expressed in the rotated reference frame. Consider

N to be the global frame and B to be the frame in which the user desires the inertia tensor. Then *rotmat* is the rotation matrix that converts a vector expressed in the basis B to a vector expressed in the basis N (i.e. vN = rotmat * vB). That is, the columns of *rotmat* are the unit vectors b_x, b_y, and b_z, each expressed in the basis given by the unit vectors n_x, n_y, n_z.

Returns

transformed [np.matrix (3,3)] If B is the frame in which the user desires the inertia tensor, this method returns ^{B}I^{H/P}, where P is the point specified by *pos*, and H is the human system.

Notes

If N is the global frame, B is the frame in which the user desires the inertia tensor, then *rotmat* = ^{N}R^{B}.

mass
Mass of the human, in units of kg.

measnames = ('Ls1L', 'Ls2L', 'Ls3L', 'Ls4L', 'Ls5L', 'Ls6L', 'Ls7L', 'Ls8L', 'Ls0p', '

print_properties(*self*, *precision=5*, *suppress=True*)
Prints human mass, center of mass, and inertia.

Parameters

precision [integer, default=5] The precision for floating point representation.

suppress [boolean, default=True] Print very small values as 0 instead of scientific notation.

Notes

See numpy.set_printoptions for more details on the optional arguments.

scale_human_by_mass(*self*, *measmass*)
Takes a measured mass and scales all densities by that mass so that the mass of the human is the same as the mesaured mass. Mass must be in units of kilograms to be consistent with the densities used.

Parameters

measmass [float] Measured mass of the human in kilograms.

segment_names = ['head-neck', 'shoulders', 'thorax', 'abdomen-pelvis', 'upper-arm', 'f

segmental_densities = {'Chandler': {'abdomen-pelvis': 853, 'foot': 1091, 'forearm':

set_CFG(*self*, *varname*, *value*)
Allows the user to set a single configuration variable in CFG. CFG is a dictionary that holds all 21 configuration variables. Then, this function validates and updates the human model with the new configuration variable.

Parameters

varname [str] Must be a valid name of a configuration variable.

value [float] New value for the configuration variable identified by varname. Units are radians. This value will be validated for joint angle limits.

set_CFG_dict(*self*, *CFG*)
Allows the user to pass an entirely new CFG dictionary with which to update the human object. Ensure that the dictionary is of the right format (ideally obtain it from a Human object with Human.CFG and modify it). After configuration is update, the segments are updated.

Parameters

CFG [dict] Stores the 21 joint angles.

update(*self*)
Redefines all solids and then calls yeadon.Human._update_segments. Called by the method yeadon.Human.scale_human_by_mass. The method is to be used in instances in which measurements change.

write_CFG(*self*, *CFGfname*)
Writes the keys and values of the self.CFG dict to a .txt file. Text file is formatted using YAML syntax.

Parameters

CFGfname [str] Filename or path to configuration output .txt file

write_meas_for_ISEG(*self*, *fname*)
Writes the values of the self.meas dict to a .txt file that is formidable as input to Yeadon's ISEG fortran code that performs similar calculations to this package. ISEG is published in Yeadon's dissertation.

Parameters

fname [str] Filename or path for ISEG .txt input file.

write_measurements(*self*, *fname*)
Writes the keys and values of the self.meas dict to a text file. Units of measurements is meters.

Parameters

fname [str] Filename or path to measurement output .txt file.

segment Module

Segment objects are used by the human module. A segment has a position, and an orientation. All constituent solids of a segment have the same orientation. That is to say that the base of the segment is at a joint in the human. The user does not interact with this module.

class yeadon.segment.**Segment**(*label*, *pos*, *rot_mat*, *solids*, *color*, *build_toward_positive_z=True*)
Bases: object

calc_properties(*self*)
Calculates the segment's center of mass with respect to the bottm center of the pelvis (Ls0) and the segment's inertia in the global frame but about the segment's center of mass.

calc_rel_properties(*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.

center_of_mass
Center of mass of the segment, a np.ndarray, in units of m, expressed in the global frame, from the bottom center of the pelvis (Ls0).

draw_mayavi(*self*, *mlabobj*)
Draws in a MayaVi window all the solids within this segment.

end_pos
Position of the center of the last (farthest from pelvis) stadium in this segment, a np.ndarray, in units of m, expressed in the global frame, from the bottom center of the pelvis (Ls0).

inertia
Inertia matrix of the segment, a np.matrix, in units of kg-m^2, about the center of mass of the human, expressed in the global frame.

mass
Mass of the segment, in units of kg.

pos
Position of the origin of the segment, a np.ndarray, in units of m, expressed in the global frame, from the bottom center of the pelvis (Ls0).

print_properties(*self*, *precision=5*, *suppress=True*)
Prints mass, center of mass (in segment and global frames), and inertia (in solid and global frames).

Parameters

precision [integer, default=5] The precision for floating point representation.

suppress [boolean, default=True] Print very small values as 0 instead of scientific notation.

Notes

See numpy.set_printoptions for more details on the optional arguments.

print_solid_properties(*self*, *precision=5*, *suppress=True*)
Calls the print_properties() member method of each of this segment's solids. See the solid class's definition of print_properties(self) for more detail.

Parameters

precision [integer, default=5] The precision for floating point representation.

suppress [boolean, default=True] Print very small values as 0 instead of scientific notation.

Notes

See numpy.set_printoptions for more details on the optional arguments.

rel_center_of_mass
Center of mass of the segment, a np.ndarray, in units of m, expressed in the frame of the segment, from the origin of the segment.

rel_inertia
Inertia matrix/dyadic of the segment, a np.matrix, in units of kg-m^2, about the center of mass of the segment, expressed in the frame of the segment.

rot_mat
Rotation matrix specifying the orientation of this segment relative to the orientation of the global frame, a np.matrix, unitless. Multiplying a vector expressed in this segment's frame with this rotation matrix on the left gives that same vector, but expressed in the global frame.

solid Module

Solid objects are used by the segment module. A solid has a position, and orientation (defined by a rotation matrix). This module also contains the class definition for stadium objects, which are used to construct StadiumSolid solids. The Solid class has two subclasses: the StadiumSolid and Semiellipsoid classes.

class yeadon.solid.**Semiellipsoid**(*label*, *density*, *baseperim*, *height*)
Bases: *yeadon.solid.Solid*

Semiellipsoid.

calc_rel_properties(*self*)
Calculates mass, relative center of mass, and relative/local inertia, according to somewhat commonly availble formulae.

draw_mayavi(*self*, *mlabobj*, *col*)
Draws the semiellipsoid in 3D using MayaVi.

Parameters

mlabobj : The MayaVi object we can draw on.

col [tuple (3,)] Color as an rgb tuple, with values between 0 and 1.

n_mesh_points = 30

class yeadon.solid.**Solid**(*label*, *density*, *height*)
Bases: object

Solid. Has two subclasses, stadiumsolid and semiellipsoid. This base class manages setting orientation, and calculating properties.

alpha = 0.5

calc_properties(*self*)
Sets the center of mass and inertia of the solid, both with respect to the fixed human frame.

center_of_mass
Center of mass of the solid, a np.ndarray of shape (3,1), in units of m, expressed in the global frame, from the bottom center of the pelvis (Ls0).

draw_mayavi(*self*, *mlabobj*, *col*)

end_pos
Position of the point on the solid farthest from the origin along the longitudinal axis of the segment, a np.ndarray of shape (3,1), in units of m, expressed in the global frame, from the bottom center of the pelvis (Ls0).

inertia
Inertia matrix of the solid, a np.matrix of shape (3,3), in units of kg-m^2, about the center of mass of the human, expressed in the global frame.

mass
Mass of the solid, a float in units of kg.

pos
Position of the origin of the solid, which is the center of the surface closest to the pelvis, a np.ndarray of shape (3,1), in units of m, expressed in the global frame, from the bottom center of the pelvis (Ls0).

print_properties(*self*, *precision=5*, *suppress=True*)
Prints mass, center of mass (in solid and global frames), and inertia (in solid and global frames).

The solid's origin is at the bottom center of the proximal stadium (or stadium closest to the pelvis, Ls0).

Parameters

precision [integer, default=5] The precision for floating point representation.

suppress [boolean, default=True] Print very small values as 0 instead of scientific notation.

Notes

See numpy.set_printoptions for more details on the optional arguments.

rel_center_of_mass
Center of mass of the solid, a np.ndarray of shape (3,1), in units of m, expressed in the frame of the solid, from the origin of the solid.

rel_inertia
Inertia matrix of the solid, a np.matrix of shape (3,3), in units of kg-m^2, about the center of mass of the solid, expressed in the frame of the solid.

set_orientation(*self*, *proximal_pos*, *rot_mat*, *build_toward_positive_z*)
Sets the position, rotation matrix of the solid, and calculates the "absolute" properties (center of mass, and inertia tensor) of the solid.

Parameters

proximal_pos [np.array (3,1)] Position of center of proximal end of solid in the absolute fixed coordinates of the human.

rot_mat [np.matrix (3,3)] Orientation of solid, with respect to the fixed coordinate system.

build_toward_positive_z [bool, optional] The order of the solids in the parent segment matters. By default they are stacked on top of each other in the segment's local +z direction. If this is set to False, then they are stacked in the local -z direction. This is done so that, for example, in the default configuration, the arms are directed down.

class yeadon.solid.**Stadium**(*label*, *inID*, *in1*, *in2=None*, *alignment='ML'*)
Bases: object

Stadium, the 2D shape.

validStadiaLabels = {'La0': 'shoulder joint centre', 'La1': 'mid-arm', 'La2': 'elbo

class yeadon.solid.**StadiumSolid**(*label*, *density*, *stadium0*, *stadium1*, *height*)
Bases: *yeadon.solid.Solid*

Stadium solid. Derived from the solid class.

calc_rel_properties(*self*)
Calculates mass, relative center of mass, and relative/local inertia, according to formulae in Appendix B of Yeadon 1990-ii. If the stadium solid is arranged anteroposteriorly, the inertia is rotated by pi/2 about the z axis.

draw_mayavi(*self*, *mlabobj*, *col*)
Draws the initial stadium in 3D using MayaVi.

Parameters

mlabobj [mayavi.soemthing] The MayaVi object we can draw on.

col [tuple (3,)] Color as an rgb tuple, with values between 0 and 1.

CHAPTER 2

Installation

The README.rst file in the source distribution of this packge contains installation instructions.

CHAPTER 3

Website navigation

- genindex
- modindex

CHAPTER 4

References

[1] M. R. Yeadon, 1990. The Simulation of Aerial Movement-ii. Mathematical Inertia Model of the Human Body. Journal of Biomechanics, 23:67-74.

[2] J. Moore, 2012. Human Control of a Bicycle. University of California, Davis.

Python Module Index

y

Index

S

U

V

W

Y