CoordSys – a Python package for handling cartesian coordinate systems

Documentation for Version 0.52

Introduction

CoordSys is is a package to handle cartesian coordinate systems and the transformation of points betweeen such systems. Main purpose is the translation between local 2D-systems and global 3D-systems. The package is partially coded in C and in pure Python.

Author: Joerg Raedler <jr@j-raedler.de>
License: LGPL
Status: should be stable, may have some memory leaks

Installation

CoordSys uses the python distutils package. Run the command:
`$ python setup.py install`
to compile and install the module. Be sure to have write permissions in pythons site-packages directory! This will install two modules on your system: cCoordSys and CoordSys.

Contents and Usage

The package currently contains two modules, a C extension module called cCoordSys and a native Python module called CoordSys.py.

CoordSys.py imports all of cCoordSys' functionality. The preferred usage is to import CoordSys.py and use the functionality of both modules. But you may import cCoordSys directly if you don't need the extensions in CoordSys.py.

Most internal errors raise the `Error` exception from the module.

Basic Example

This creates a coordiante system with an offset `o` and three vectors `u`,`v` and `w`. A list of global points `glo` is transformed to local coordiantes `loc` and back to global `gl2`.

```
>>> from CoordSys import CoordSys
>>> o = (3,4,5)
>>> u = (0, -1, 0)
>>> v = (0, 0, 1)
>>> w = (1, 0, 0)
>>> cs = CoordSys(o, u, v, w)
>>> glo = ((0,1,2), (3,8,7), (-1,2,3)
>>> loc = cs.toLocal(glo)
>>> gl2 = cs.toGlobal(loc)
```

Module cCoordSys: core CoordSys objects

cCoordSys contains the following objects:

Symbol	***Description***
`CoordSys`	CoordSys type to hold a number of contours
`CoordSysType`	Type object for CoordSys
`version`	Version string of the package
`Error`	the exception raised when methods or operations fail

CoordSys objects

In this library a coordinate system is a collection of four vectors based on the global system. An offset `o` is the origin of the system. The vectors `u`, `v` and `w` are the axes of the system. When converting to a local 2D system, there's an optional check if a point fits on the 2d plane. You may activate the check by setting a tolerance (max. distance between point and plane). By setting the tolerance to `None` you deactivate the check which will simply drop the `w` component. If a check failes, `Error` is raised.

Be carefull: The 2D system will usually be built from the first two points and the last point in the list. If one of these points is a little bit inaccurate, your 2D-plane may be completely wrong for distant points!

CoordSys methods

Method	***Arguments***	***Returns***	***Description***
`CoordSys(|o, u, v, w)`	No arguments or four sequences of three numbers each	CoordSys object	Constructor: Create a new CoordSys object.
`toGlobal(plist)`	- pointlist – sequence of points (2D or 3D)	pointlist – sequence of 3D points	Convert to global system
`toLocal(plist)`	- pointlist – sequence of 3D points	pointlist – sequence of 3D points	Convert to global system
`toLocal2D(plist)`	- pointlist – sequence of 3D points	pointlist – sequence of 2D points	Convert to global system without the w-component
`setTol2D(tol)`	- tolerance value – float or - None	None	Set tolerance for 3D-2D conversion
`getTol2D()`	- None	float value	Get tolerance
`find2D(plist |, n=1)`	- pointlist – sequence of 3D points - a boolean value	None	Adjust system for a 2D-plane by using the points. If n is set (which is default) the vectors u,v and w will be normalized.
`offset()`	- None	a sequence of 3 floats	return the o vector
`matrix()`	- None	a 3-sequence of 3-sequences of floats (3x3 matrix)	return the transformation matrix built from u, v and w
`o()`	- None	a sequence of 3 floats	return the o vector (alias for offset())
`u()`	- None	a sequence of 3 floats	return the u vector
`v()`	- None	a sequence of 3 floats	return the v vector
`w()`	- None	a sequence of 3 floats	return the w vector

Module CoordSys.py : additional methods

- `planeCheck(pointlist, tolerance)`
  checks if the 3D points are on a plane. This will build 2D system, set tolerance and convert all points to the system. If `Error` is raised, 0 is returned, 1 otherwise. The 2D system is build as described above. If the first two or the last point are inaccurate, the system may be completely wrong and give bad results!