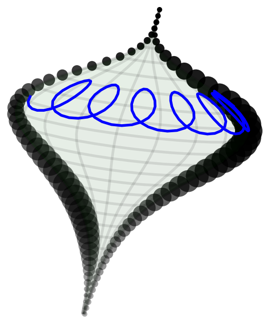

GyroSoft Simulation Documentation

Release 0.0.1

TL

January 19, 2015

Contents

Contents:

This document shows all the functions used in the python program to calculate the equations of motion of the top, the parameters users can change, and the 3D figure and animation generating functions.

1 Funtional Descriptions of equation of motion of top

Module of the Rigid Body class.

Created on Tue Nov 11 16:05:05 2014

@author: whymranderson.blogspot.tw

There are two global functions CK and euler2space.

`RGCordTransV13.`**`CK`**`(`*rotvec*`)`

Cayley-Klein parameters. Important: the built rotation matrix is with its couterclockwise active sense. This is basically the Rodriguez rotation formula in a matric form.

`RGCordTransV13.`**`euler2space`**`(`*eulerW*`)`

Return Euler angles transformation matrix from lab frame to time t frame.

Matrix takes on passive right-hand sense. The list of input argument w is w(0):phi, w(1):phi_dot, w(2):theta, w(3):theta_dot, w(4):psi

---

Functions of RigidBodyOjbect class followed by parameters of top that can be used by the user in the program:

class `RGCordTransV13.`**`RigidBodyObject`**

Rigid body class

EulerDCMiter()

Solve Euler equations directly with a integration of DCM method.

This is the A method. Also this method use w(t_{i+1}) in rotation vector approximation. See manuscript.

EulerDCMiter_wt_i()

‘This is A method with w(t_{i}) approximation.

GenerateDependentVariables()

Initialize arrays and calculate needed parameters

HasbunEulerEquationODEsolve()

Numerically solve euler angles euler equations using Python ode library.

This is rewritten from Prof Hasbun’s matlab code in his book “Classical Mechanics with MATLAB Applications” The list of Euler_angles_hasbun = w(0):phi, w(1):phi_dot, w(2):theta, w(3):theta_dot, w(4):psi

directDCMiter()

direct DCM iteration from body anguler velocity. This is method C.

w_b are calculated from method B

eulerW2bodyW()

Convert Hasbun’s euler angles solution to body angular velocity

The euler_angles_hasbun array has size N+1-1

setcase(*case*, **arg*)

Choose from four classical nutation and precession motions or set customary initial condition

topEOM(*wi*, *torquei*)

EOM of a symmetric top

topRK(*wi*, *torquei*)

4th order Rugge Kutta numeric solver

uniformprecesscal()

return the initial condition required to generate a uniform precession top.

Calculation base on the parameters and the setup of the top. The formula is in Goldstein’s book. Also this formula will have a singularity at theta = 90 degrees.

Ix = 0.0005768750000000001

x moment of inertia

Iy = 0.0005768750000000001

y moment of inertia

Iz = 9.375000000000002e-05

z moment of inertia

L = 0.005

width of disk in m

M = 0.3

mass of top in kg

R = 0.025

top is a disk with radius R in meters

arm = 0.01

location of center of mass of top from origin in meters

classical_case = 1

selection of four typical gyroscopic motions: 1,2,3,4

`counter_weight` **= 0.05**
    other mass used to balance the gyro

`counter_weight_location_from_origin` **= 0.1**
    location from origin

`freq` **= 20.0**
    top revolution speed in hertz, along symmetric axis

`g` **= 9.8**
    gravity m/s^2

`orien` **= array([-1.04719755, 0. , 0.])**
    starting orientation vector of z' of top from lab xyz

`samplerate` **= 2000**
    rate of iteration in Hz

`t0` **= 0.0**
    start of time zero

`tn` **= 1.5**
    end of simulation time, take longer time if tn > 10 seconds

---

The following parameters are used in command like `Drawoption['A_axes'] = True` to make the belonging physical observable visible in a 3D animation.

class `RGCordTransV13.`**`RigidBodyObject`**

`GenerateDependentVariables``()`

`DrawOption`

‘A_axes’: False,

‘B_axes’: False,

‘C_axes’: False,

‘A_z_axis_trace’: False,

‘B_z_axis_trace’: False,

‘A_Angular Momentum Trace’ : False,

‘B_Angular Momentum Trace’ : False,

‘Angular Velocity Trace’ : False,

‘A_Angular Momentum Vec’ : False,

‘B_Angular Momentum Vec’ : False,

‘A_Angular Velocity Vec’ : False,

‘A_square’: False

2 3D plotting and animation functions

Plot and animation functions/environment setup for Rigid Body class

Created on Tue Nov 11 16:05:05 2014

@author: whymranderson.blogspot.tw

`RBPlotFunctionV3.`**`AnimationFrameSetUp`**`(`*RBO*, *ax*, *plt*`)`
: plot the first frame of animation

`RBPlotFunctionV3.`**`CalDiffInDeg`**`(`*cordvec*`)`
: Calculate and return angle difference between A and B method's 111 unit vectors

`RBPlotFunctionV3.`**`CalDiffInDegBC`**`(`*cordvec*`)`
: Calculate and return angle difference between B and C method's 111 unit vectors

`RBPlotFunctionV3.`**`get111norm`**`(`*xyzaxes*`)`
: Return array of body (1,1,1) unit vector as a function of time N in size (N,3)

`RBPlotFunctionV3.`**`getPlotAnimationLines`**`(`*RBO*`)`
: Construct and return the line animation data sequence in a rigid body object for 3D animation

`RBPlotFunctionV3.`**`plot_back`**`(`*axes*`)`
: plot the lucid sphere, back part

`RBPlotFunctionV3.`**`plot_front`**`(`*axes*`)`
: plot the lucid sphere, front part

`RBPlotFunctionV3.`**`update_line_new`**`(`*x*, **arg*`)`
: line animation function, will be called by functionanimation, take multiple line data sequences as arguments