FrisPy

Tom McClintock

Nov 16, 2021

CONTENTS:

CHAPTER

ONE

FRISPY

Documentation for `FrisPy` package can be found here on RTD.

This repository contains a physical model for a flying disc. Using this code, one can simulate trajectories of discs with varying initial conditions, while also changing the underlying physical modlel. This is useful for analyzing the mechanics of a disc in terms of its design, as well as creating simulated throws for things like disc launchers or other helpful tools.

This is a pure Python rebuild of the old FrisPy code, which included a version of the integrator written in C for speed. To obtain a fast version of the modeling code, either roll back to an old version or check out the Frisbee_Simulator repository.

1.1 Installation

The easiest way to install this package is with `pip`. The PyPI package can be viewed here.

```
pip install frispy
```

To install from source, there are other steps involved. First, you must obtain the code from Github. If you have git installed you can clone the repository from the command line:

```
git clone https://github.com/tmcclintock/FrisPy.git
```

or with the GitHub Desktop application. Once you have the code, change into the directory and proceed.

Note, the only hard requirements for this package are `python>=3.6`, `numpy`, `scipy`, and `matplotlib` (plotting only). Note that this package uses the relatively recent scipy.integrate.solve_ivp method, which may not exist in older versions of `scipy`. If you have these three packages, you can install this package with the `setup.py` file without worrying about creating an environment.

1.2 From an Anaconda environment

The preferred method of installation is with anaconda You can install all the requirements into a compatible environment called `frispy` by running the following command:

```
conda env create -f environment.yml
```

You can then install the package the usual way

```
python setup.py install
```

You can also use pip to install the requirements from the requirements.txt file by running:

```
pip install -r requirements.txt
```

Then follow this by using the setup.py file to install.

1.3 Testing

Verify your installation by running:

```
pytest
```

Please report any problems you encounter on the issues page. Thank you!

1.4 Running

Check out example.py to see how to run and view results. In words, you create a disc and compute its trajectory.

```
from frispy import Disc

disc = Disc()
result = disc.compute_trajectory()
times = result.times
x, y, z = result.x, result.y, result.z
```

Once you have a trajectory, you can use that to create visualizations. For instance, to plot the height of the disc against one of its horizontal coordintes (x), you can run:

```
import matplotlib.pyplot as plt

plt.plot(x, z)
plt.show()
```

1.5 Soon

There are some big upgrades on the horizon! Stay tuned for:

- animated trajectories
- documentation
- example Jupyter notebooks
- plotting routines

CHAPTER

TWO

API REFERENCE

This page contains auto-generated API reference documentation[1].

2.1 frispy

2.1.1 Submodules

frispy.disc

Module Contents

Classes

Disc	Flying spinning disc object. The disc object contains only physical
Result	A namedtuple subclass that contains the coordinate variables

class frispy.disc.**Disc**(*model: frispy.model.Model = Model()*, *eom: Optional[frispy.equations_of_motion.EOM] = None*, ***kwargs*)

Flying spinning disc object. The disc object contains only physical parameters of the disc and environment that it exists (e.g. gravitational acceleration and air density). Note that the default area, mass, and inertial moments are for Discraft Ultrastars (175 grams or 0.175 kg).

All masses are kg, lengths are meters (m) and times are seconds (s). That is, these files all use *mks* units. Angular units use radians (rad), and angular velocities are in rad/s.

Parameters

- **model** (Model, *optional*) –
- **eom** (EOM, *optional*) – the equations of motion
- **kwargs** – keyword arguments of a numeric type to specify the initial conditions of the disc. For example x=3 or vz=10..

_default_initial_conditions

compute_trajectory(*self*, *flight_time: float = 3.0*, *return_scipy_results: bool = False*, ***kwargs*)

Call the differential equation solver to compute the trajectory. The kinematic variables and timesteps are

[1] Created with sphinx-autoapi

saved as the *current_trajectory* attribute, which is a dictionary, which is also returned by this function.

See the scipy docs for more information on the solver.

> **Warning:** You cannot pass a *flight_time* if *t_span* is a key in *solver_args*.

Parameters

- **flight_time** (*float, optional*) – time in seconds that the simulation will run over. Default is 3 seconds.
- **return_scipy_results** (*bool, optional*) – Default is *False*. Flag to indicate whether to return the full results object of the solver. See the scipy docs for more information.
- **kwargs** – extra keyword arguments to pass to the `scipy.integrate.solver_ivp()`

reset_initial_conditions(*self*) → None
Set the initial_conditions of the disc to the default and clear the trajectory.

set_default_initial_conditions(*self*, ***kwargs*) → None

property **environment**(*self*) → *frispy.environment.Environment*

property **trajectory_object**(*self*) → *frispy.trajectory.Trajectory*

property **coordinate_names**(*self*) → List[str]
Names of the kinematic variables

class frispy.disc.**Result**
Bases: `namedtuple('Result', list(Disc._default_initial_conditions.keys())+['times'])`

A `namedtuple` subclass that contains the coordinate variables and a `times` attribute. One can reference the variables in the result as an attribute `result.x`.

frispy.environment

The `Environment` object.

Module Contents

Classes

Environment	The environment in which the disc is flying in. This object contains

class frispy.environment.**Environment**
Bases: `NamedTuple`

The environment in which the disc is flying in. This object contains information on the magnitude and direction of gravity, properties of the wind, and also intrinsic properties of the disc such as its area and mass.

Parameters

- **air_density** (*float*) – default is 1.225 kg/m^3
- **area** (*float*) – default is 0.057 m^2

- **g** (*float*) – default is 9.81 m/s^2; gravitational acceleration on Earth
- **I_zz** (*float*) – default is 0.002352 kg*m^2; z-axis moment of inertia
- **I_xx** (*float*) – default is 0.001219 kg*m^2; x and y-axis moments of inertia (i.e. is the same as I_yy and the cross components I_xy)
- **mass** (*float*) – defualt is 0.175 kg

air_density :float = 1.225

area :float = 0.058556

g :float = 9.81

I_zz :float = 0.002352

I_xx :float = 0.001219

mass :float = 0.175

property grav_unit_vector(*self*) → numpy.ndarray
Gravitational direction.

property diameter(*self*) → float
Disc diameter.

frispy.equations_of_motion

Module Contents

Classes

EOM	EOM is short for "equations of motion" is used to run the ODE solver

class frispy.equations_of_motion.**EOM**(*environment: frispy.environment.Environment = Environment()*, *model: frispy.model.Model = Model()*, *trajectory: frispy.trajectory.Trajectory = Trajectory()*)

EOM is short for "equations of motion" is used to run the ODE solver from *scipy*. It takes in a model for the disc, the trajectory object, the environment, and implements the functions for calculating forces and torques.

compute_forces(*self*, *phi: float*, *theta: float*, *velocity: numpy.ndarray*, *ang_velocity: numpy.ndarray*) → Dict[str, Union[float, numpy.ndarray, Dict[str, numpy.ndarray]]]
Compute the lift, drag, and gravitational forces on the disc.

compute_torques(*self*, *velocity: numpy.ndarray*, *ang_velocity: numpy.ndarray*, *res: Dict[str, Union[float, numpy.ndarray, Dict[str, numpy.ndarray]]]*) → Dict[str, Union[float, numpy.ndarray, Dict[str, numpy.ndarray]]]
Compute the torque around each principle axis.

compute_derivatives(*self*, *time: float*, *coordinates: numpy.ndarray*) → numpy.ndarray
Right hand side of the ordinary differential equations. This is supplied to `scipy.integrate.solve_ivp()`. See this page for more information about its *fun* argument.

Parameters

- **time** (*float*) – instantanious time of the system
- **coordinates** (*np.ndarray*) – kinematic variables of the disc

Returns derivatives of all coordinates

frispy.model

Physical model for the forces and torques on a disc.

Module Contents

Classes

Model	Coefficient model for a disc. Holds all of the aerodynamic

class frispy.model.**Model**

Coefficient model for a disc. Holds all of the aerodynamic parameters coupling the kinematic variables (spins and angles) to the force magnitudes.

PL0 :float = 0.33

PLa :float = 1.9

PD0 :float = 0.18

PDa :float = 0.69

PTxwx :float = 0.43

PTxwz :float

PTy0 :float

PTya :float

PTywy :float

PTzwz :float

alpha_0 :float

C_lift(*self*, *alpha: float*) → float

Lift force scale factor. Linear in the angle of attack (*alpha*).

Parameters **alpha** (*float*) – angle of attack in radians

Returns (float) lift force scale factor

C_drag(*self*, *alpha: float*) → float

Drag force scale factor. Quadratic in the angle of attack (*alpha*).

Parameters **alpha** (*float*) – angle of attack in radians

Returns (float) drag force scale factor

C_x(*self*, *wx: float*, *wz: float*) → float

'x'-torque scale factor. Linearly additive in the 'z' angular velocity (*w_z*) and the 'x' angular velocity (*w_x*).

Parameters

- **wx** (*float*) – 'x' angular velocity in radians per second
- **wz** (*float*) – 'z' angular velocity in radians per second

Returns (float) 'x'-torque scale factor

C_y(*self*, *alpha: float*, *wy: float*) → float

'y'-torque scale factor. Linearly additive in the 'y' angular velocity (*w_y*) and the angle of attack (*alpha*).

Parameters

- **alpha** (*float*) – angle of attack in radians
- **wy** (*float*) – 'y' angular velocity in radians per second

Returns (float) 'y'-torque scale factor

C_z(*self*, *wz: float*) → float

'z'-torque scale factor. Linear in the 'z' angular velocity (*w_z*).

Parameters **wz** (*float*) – 'z' angular velocity in radians per second

Returns (float) 'z'-torque scale factor

frispy.trajectory

The `Trajectory` is the interface to the differential equation solver for the disc trajectory.

Module Contents

Classes

Trajectory	Class for computing the disc flight trajectory. Takes initial values

Functions

rotation_matrix(phi: float, theta: float) → numpy.ndarray	Compute the (partial) rotation matrix that transforms from the

`frispy.trajectory.`**rotation_matrix**(*phi: float*, *theta: float*) → numpy.ndarray

Compute the (partial) rotation matrix that transforms from the lab frame to the disc frame. Note that because of azimuthal symmetry, the azimuthal angle (*gamma*) is not used.

class `frispy.trajectory.`**Trajectory**

Class for computing the disc flight trajectory. Takes initial values and interfaces with an ODE solver.

Units are meters [m] for length, kilograms [kg] for mass, seconds [s] for time, and radians [rad] for angles.

Parameters

- **x** (*float*) – horizontal position; default is 0 m
- **y** (*float*) – horizontal position; default is 0 m
- **z** (*float*) – vertical position; default is 1 m
- **vx** (*float*) – x-velocity; default is 10 m/s
- **vy** (*float*) – y-velocity; default is 0 m/s

- **vz** (*float*) – z-velocity; default is 0 m/s
- **phi** (*float*) – 1st Euler angle (pitch); default is 0 rad
- **theta** (*float*) – 2nd Euler angle (roll); default is 0 rad
- **gamma** (*float*) – 3rd Euler angle (spin); default is 0 rad
- **phidot** (*float*) – phi angular velocity; default is 0 rad/s
- **thetadot** (*float*) – theta angular velocity; default is 0 rad/s
- **gammadot** (*float*) – gamma angular velocity; default is 50 rad/s

x :float = 0

y :float = 0

z :float = 1

vx :float = 10

vy :float = 0

vz :float = 0

phi :float = 0

theta :float = 0

gamma :float = 0

phidot :float = 0

thetadot :float = 0

gammadot :float = 50

__post_init__(*self*)

reset(*self*) → None

property velocity(*self*) → numpy.ndarray

property angular_velocity(*self*) → numpy.ndarray

derived_quantities(*self*) → Dict[str, Union[float, numpy.ndarray, Dict[str, numpy.ndarray]]]
Compute intermediate quantities on the way to computing the time derivatives of the kinematic variables.

frispy.wind

The `Wind` class handles the wind, which is a real-valued time-dependent vector field that influences the flight of the disc.

Module Contents

Classes

Wind	Abstract class to handle different types of wind. These can include
NoWind	No wind.
ConstantWind	The wind is uniform in position and constant in time.

class frispy.wind.**Wind**

Bases: abc.ABC

Abstract class to handle different types of wind. These can include steady, laminar flow winds or swirling winds. Winds can have a time dependence to mimic "gusts".

abstract get_wind(*self*, *t: Optional[Union[float, int, numpy.ndarray]]*, *position: Optional[Union[List, numpy.ndarray]]*) → numpy.ndarray

Obtain a length 3 vector of the wind at time *t*.

class frispy.wind.**NoWind**

Bases: *Wind*

No wind.

get_wind(*self*, **args*) → numpy.ndarray

All components are zero.

class frispy.wind.**ConstantWind**(*wind_vector: Optional[numpy.ndarray] = None*)

Bases: *Wind*

The wind is uniform in position and constant in time.

get_wind(*self*, **args*) → numpy.ndarray

Obtain a length 3 vector of the wind at time *t*.

2.1.2 Package Contents

Classes

Disc	Flying spinning disc object. The disc object contains only physical
Environment	The environment in which the disc is flying in. This object contains
EOM	EOM is short for "equations of motion" is used to run the ODE solver
Model	Coefficient model for a disc. Holds all of the aerodynamic
Trajectory	Class for computing the disc flight trajectory. Takes initial values

Attributes

__author__
__version__
__docs__

class frispy.**Disc**(*model:* frispy.model.Model *= Model()*, *eom: Optional[*frispy.equations_of_motion.EOM*] = None*, ***kwargs*)

Flying spinning disc object. The disc object contains only physical parameters of the disc and environment that it exists (e.g. gravitational acceleration and air density). Note that the default area, mass, and inertial moments are for Discraft Ultrastars (175 grams or 0.175 kg).

All masses are kg, lengths are meters (m) and times are seconds (s). That is, these files all use *mks* units. Angular units use radians (rad), and angular velocities are in rad/s.

Parameters

- **model** (`Model`, *optional*) –
- **eom** (`EOM`, *optional*) – the equations of motion
- **kwargs** – keyword arguments of a numeric type to specify the initial conditions of the disc. For example `x=3` or `vz=10.`.

_default_initial_conditions

compute_trajectory(*self*, *flight_time: float = 3.0*, *return_scipy_results: bool = False*, ***kwargs*)

Call the differential equation solver to compute the trajectory. The kinematic variables and timesteps are saved as the *current_trajectory* attribute, which is a dictionary, which is also returned by this function.

See the scipy docs for more information on the solver.

> **Warning:** You cannot pass a *flight_time* if *t_span* is a key in *solver_args*.

Parameters

- **flight_time** (*float, optional*) – time in seconds that the simulation will run over. Default is 3 seconds.
- **return_scipy_results** (*bool, optional*) – Default is *False*. Flag to indicate whether to return the full results object of the solver. See the scipy docs for more information.
- **kwargs** – extra keyword arguments to pass to the `scipy.integrate.solver_ivp()`

reset_initial_conditions(*self*) → None

Set the initial_conditions of the disc to the default and clear the trajectory.

set_default_initial_conditions(*self*, ***kwargs*) → None

property environment(*self*) → *frispy.environment.Environment*

property trajectory_object(*self*) → *frispy.trajectory.Trajectory*

property coordinate_names(*self*) → List[str]

Names of the kinematic variables

class frispy.**Environment**

Bases: `NamedTuple`

The environment in which the disc is flying in. This object contains information on the magnitude and direction of gravity, properties of the wind, and also intrinsic properties of the disc such as its area and mass.

Parameters

- **air_density** (*float*) – default is 1.225 kg/m^3
- **area** (*float*) – default is 0.057 m^2
- **g** (*float*) – default is 9.81 m/s^2; gravitational acceleration on Earth
- **I_zz** (*float*) – default is 0.002352 kg*m^2; z-axis moment of inertia
- **I_xx** (*float*) – default is 0.001219 kg*m^2; x and y-axis moments of inertia (i.e. is the same as I_yy and the cross components I_xy)

- **mass** (*float*) – defualt is 0.175 kg

air_density :float = 1.225

area :float = 0.058556

g :float = 9.81

I_zz :float = 0.002352

I_xx :float = 0.001219

mass :float = 0.175

property **grav_unit_vector**(*self*) → numpy.ndarray
Gravitational direction.

property **diameter**(*self*) → float
Disc diameter.

class frispy.**EOM**(*environment:* frispy.environment.Environment *= Environment()*, *model:* frispy.model.Model *= Model()*, *trajectory:* frispy.trajectory.Trajectory *= Trajectory()*)
EOM is short for "equations of motion" is used to run the ODE solver from *scipy*. It takes in a model for the disc, the trajectory object, the environment, and implements the functions for calculating forces and torques.

compute_forces(*self*, *phi: float*, *theta: float*, *velocity: numpy.ndarray*, *ang_velocity: numpy.ndarray*) → Dict[str, Union[float, numpy.ndarray, Dict[str, numpy.ndarray]]]
Compute the lift, drag, and gravitational forces on the disc.

compute_torques(*self*, *velocity: numpy.ndarray*, *ang_velocity: numpy.ndarray*, *res: Dict[str, Union[float, numpy.ndarray, Dict[str, numpy.ndarray]]]*) → Dict[str, Union[float, numpy.ndarray, Dict[str, numpy.ndarray]]]
Compute the torque around each principle axis.

compute_derivatives(*self*, *time: float*, *coordinates: numpy.ndarray*) → numpy.ndarray
Right hand side of the ordinary differential equations. This is supplied to `scipy.integrate.solve_ivp()`. See this page for more information about its *fun* argument.

Parameters

- **time** (*float*) – instantanious time of the system
- **coordinates** (*np.ndarray*) – kinematic variables of the disc

Returns derivatives of all coordinates

class frispy.**Model**
Coefficient model for a disc. Holds all of the aerodynamic parameters coupling the kinematic variables (spins and angles) to the force magnitudes.

PL0 :float = 0.33

PLa :float = 1.9

PD0 :float = 0.18

PDa :float = 0.69

PTxwx :float = 0.43

PTxwz :float

PTy0 :float

PTya :float

PTywy :float

PTzwz :float

alpha_0 :float

C_lift(*self*, *alpha: float*) → float
Lift force scale factor. Linear in the angle of attack (*alpha*).

Parameters **alpha** (*float*) – angle of attack in radians

Returns (float) lift force scale factor

C_drag(*self*, *alpha: float*) → float
Drag force scale factor. Quadratic in the angle of attack (*alpha*).

Parameters **alpha** (*float*) – angle of attack in radians

Returns (float) drag force scale factor

C_x(*self*, *wx: float*, *wz: float*) → float
'x'-torque scale factor. Linearly additive in the 'z' angular velocity (*w_z*) and the 'x' angular velocity (*w_x*).

Parameters

- **wx** (*float*) – 'x' angular velocity in radians per second
- **wz** (*float*) – 'z' angular velocity in radians per second

Returns (float) 'x'-torque scale factor

C_y(*self*, *alpha: float*, *wy: float*) → float
'y'-torque scale factor. Linearly additive in the 'y' angular velocity (*w_y*) and the angle of attack (*alpha*).

Parameters

- **alpha** (*float*) – angle of attack in radians
- **wy** (*float*) – 'y' angular velocity in radians per second

Returns (float) 'y'-torque scale factor

C_z(*self*, *wz: float*) → float
'z'-torque scale factor. Linear in the 'z' angular velocity (*w_z*).

Parameters **wz** (*float*) – 'z' angular velocity in radians per second

Returns (float) 'z'-torque scale factor

class frispy.**Trajectory**
Class for computing the disc flight trajectory. Takes initial values and interfaces with an ODE solver.

Units are meters [m] for length, kilograms [kg] for mass, seconds [s] for time, and radians [rad] for angles.

Parameters

- **x** (*float*) – horizontal position; default is 0 m
- **y** (*float*) – horizontal position; default is 0 m
- **z** (*float*) – vertical position; default is 1 m
- **vx** (*float*) – x-velocity; default is 10 m/s
- **vy** (*float*) – y-velocity; default is 0 m/s
- **vz** (*float*) – z-velocity; default is 0 m/s
- **phi** (*float*) – 1st Euler angle (pitch); default is 0 rad
- **theta** (*float*) – 2nd Euler angle (roll); default is 0 rad

- **gamma** (*float*) – 3rd Euler angle (spin); default is 0 rad
- **phidot** (*float*) – phi angular velocity; default is 0 rad/s
- **thetadot** (*float*) – theta angular velocity; default is 0 rad/s
- **gammadot** (*float*) – gamma angular velocity; default is 50 rad/s

x :float = 0

y :float = 0

z :float = 1

vx :float = 10

vy :float = 0

vz :float = 0

phi :float = 0

theta :float = 0

gamma :float = 0

phidot :float = 0

thetadot :float = 0

gammadot :float = 50

__post_init__(*self*)

reset(*self*) → None

property velocity(*self*) → numpy.ndarray

property angular_velocity(*self*) → numpy.ndarray

derived_quantities(*self*) → Dict[str, Union[float, numpy.ndarray, Dict[str, numpy.ndarray]]]
Compute intermediate quantities on the way to computing the time derivatives of the kinematic variables.

frispy.**__author__ = Tom McClintock thmsmcclintock@gmail.com**

frispy.**__version__ = 1.0.4**

frispy.**__docs__ = Simulates flying spinning discs.**

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

INDEX

Symbols

A

C

D

E

F

G

I

M

N

P

R

S

T

V

W

X

Y

Z