Autonomous Vehicle Simulation (AVS) Laboratory, University of Colorado

Basilisk Technical Memorandum

Document ID: Basilisk-Atmosphere

ATMOSPHERE

Prepared by	A. Harris

Status: Released
Scope/Contents
The `Atmosphere` class used to calculate temperature / density above a body using multiple models. This class is used to hold relevant atmospheric properties and to compute the density for a given set of spacecraft relative to a specified planet. Planetary parameters, including position and input message, are settable by the user. Internal support is provided for Venus, Earth, and Mars. In a given simulation, each planet of interest should have only one Atmosphere model associated with it linked to the spacecraft in orbit about that body.

Rev	Change Description	By	Date
1.0	Initial release	A. Harris	02-26-2019

Contents

---

1 Model Description

The purpose of this module is to implement neutral atmospheric density and temperature models. Atmosphere represents a single interface to a variety of atmospheric models within the BSK framework. These are briefly summarized here for reference.

1.1 Exponential Atmosphere

Under the assumption of an isothermal atmosphere, an expression for atmospheric density can be readily arrived at by combining the equations of hydrostatic equilibrium with the ideal gas law and integrating, yielding:

$$\rho = \rho_0 e^{\frac{-h}{h_0}} \tag{1}$$

where ρ_0 is the density at the planet's surface, h is the altitude, and h_0 is the atmospheric scale height derived from the weighted-average behavior of the species that make up the atmosphere. A list of these parameters for atmospheric bodies in the solar system is available from NASA here. For more detail, see Reference 1.

1.2 NRLMSISE-00

NRLMSISE-00 is an empircally derived neutral density and temperature model maintained by the Naval Research Laboratory. Basilisk uses the C implementation of NRLMSISE-00 published by Dr. Dominik Brodowski.

At present, this functionality is not implemented.

2 Module Functions

This module will:

- **Compute atmospheric density and temperature**: Each of the provided models is fundamentally intended to compute the neutral atmospheric density and temperature for a spacecraft relative to a body. These parameters are stored in the AtmoPropsSimMsg struct. Supporting parameters needed by each model, such as planet-relative position, are also computed.
- **Communicate neutral density and temperature**: This module interfaces with modules that subscribe to neutral density messages via the messaging system.
- **Subscribe to model-relevant information:** Each provided atmospheric model requires different input information to operate, such as current space weather conditions and spacecraft positions. This module automatically attempts to subscribe to the relevant messages for a specified model.
- **Support for multiple spacecraft and model types** Only one Atmosphere module is required for each planet, and can support an arbitrary number of spacecraft. Output messages for individual spacecraft are automatically named based on the environment type.

3 Module Assumptions and Limitations

Individual atmospheric models are complex and have their own assumptions. At present, all non-exponential models are Earth-specific. For details about tradeoffs in atmospheric modeling, the reader is pointed to 1.

4 Test Description and Success Criteria

This section describes the specific unit tests conducted on this module.

4.1 General Functionality

4.1.1 setEnvType

This test verifies that the module is able adjust its internal state to each of the implemented atmospheric models.

4.1.2 setEpoch

This test verifies that the user can set the initial date ("epoch") arbitrarily.

4.1.3 addSpacecraftToModel

This test verifies that the user can both add additional spacecraft to the module. This is accomplished by checking the number of input and output messages of the module after adding multiple spacecraft.

4.2 Model-Specific Tests

4.2.1 testExpAtmo

This model runs a section of an orbit and verifies that the exponential atmosphere model both correctly calculates the orbit altitude and the atmospheric density against a Python implementation of the model.

5 Test Parameters

Test and simulation parameters and inputs go here. Basically, describe your test in the section above, but put any specific numbers or inputs to the tests in this section.

The unit test verify that the module output guidance message vectors match expected values.

Table 2: Error tolerance for each test.

Output Value Tested	Tolerated Error
length(atmo.scStateInMsgNames)	0 (int)
length(atmo.envOutMsgNames)	0 (int)
atmo.envType	0 (string)
neutralDensity	1e-13
altitude	1e-13

6 Test Results

The results of the unit test should be included in the documentation. The results can be discussed verbally, but also included as tables and figures.

All of the tests passed:

Table 3: Test results

Check	Pass/Fail
1	PASSED

7 User Guide

7.1 General Module Setup

This section outlines the steps needed to add an Atmosphere module to a sim.

First, the atmosphere must be imported and initialized:

```
from Basilisk.simulation import atmosphere

newAtmo = atmosphere.Atmosphere()
```

Next, the desired model type must be set by calling "setEnvType":

```
atmoTaskName = "atmosphere"
newAtmo.ModelTag = "ExpAtmo"
newAtmo.setEnvType("exponential")
```

The model can then be added to a task like other simModels. Each Atmosphere calculates atmospheric parameters based on the output state messages for a set of spacecraft; to add spacecraft to the model the addScToModel method is invoked:

```
scObject = spacecraftPlus.SpacecraftPlus()
scObject.ModelTag = "spacecraftBody"
newAtmo.addSpacecraftToModel(scObject.scStateOutMsgName)
```

Finally, the planet state message name can be set by directly adjusting that attribute of the class:

```
newAtmo.planetPosInMsgName = "PlanetSPICEMsgName"
```

If SPICE is not being used, the planet is assumed to reside at the origin.

7.2 Exponential atmosphere user guide

If the module's envType is set to "exponential", the parameters of the exponential atmosphere can be set by calling

```
newAtmo.exponentialParams.baseDensity = baseDensityValue
newAtmo.exponentialParams.scaleHeight = scaleHeightValue
newAtmo.exponentialParams.planetRadius = planetRadiusValue
```

REFERENCES

[1] David Vallado. *Fundamentals of Astrodynamics and Applications*. Microcosm press, 4 edition, 2013.