JSC-61777-verif/SIM_dyncomp

JSC Engineering Orbital Dynamics Dynamics Comparison Simulation

Simulation and Graphics Branch (ER7)
Software, Robotics, and Simulation Division
Engineering Directorate

Package Release JEOD v5.0

Document Revision 3.0
July 2022

National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas

JSC-61777-verif/SIM_dyncomp

JSC Engineering Orbital Dynamics
Dynamics Comparison Simulation

Document Revision 3.0
July 2022

A.A. Jackson and G. Turner

Simulation and Graphics Branch (ER7)
Software, Robotics, and Simulation Division
Engineering Directorate

National Aeronautics and Space Administration
Lyndon B. Johnson Space Center
Houston, Texas

Abstract

Regression testing of the JSC Engineering Dynamics (JEOD) software is required in order to assure the user and any clients regarding the validity of the results of the simulation. There are many levels of software verification; the focus of Dynamics Comparison Simulation is the establishment of set of tests using the numerical output of the simulation compared against other simulation data and simple analytic unit tests. The goal of this subset of validation testing of JEOD is a demonstration that the software meets the purposes and requirements of the over all project when new versions of the software is issued.

Contents

Chapter 1

Introduction

1.1 Introduction

After any revision of the JEOD software there is a process of regression testing. Each version of JEOD is verified and validated in two ways. All releases are tested for accuracy against trajectories of actual vehicles. They are also regression tested by comparing the results of individual model vetification simulations, and by running combined simulations such as Dynamics Comparison Simulation.

The present version, as it becomes available, is tested against the last version and checked for differences. This means all code,test runs, and documents are checked against the previous JEOD as a truth model. Any differences between code, documents and test runs for models that did not change have to be explained or resolved. Once resolution has been achieved, the data files for regression testing are archived in house in the JEOD directories. Any change – whether it be a code change, model addition, document changed or test run addition – is documented in the top level JEOD documents. These documents are made available, along with version notes at the time of JEOD official release.

The documentation for SIM_dyncomp for this release is contained in the following document: A Process for Comparing Dynamics of Distributed Space Systems Simulations[1]. It is hyperlinked here: simpaper

The validation of these simulations are explained in the paper. Note the validation data consisted of independent simulation to simulation comparisons and simple analytical models what could be computed independent of JEOD.

1.2 Document History

Author	Date	Revision	Description
A.A. Jackson	DEC 2009	1.0	Initial Version
A.A. Jackson	SEP 2010	2.0	Revision
G. Turner	JUN 2018	3.0	Replaced S_define with new version

1.3 SIM_dyncomp S_define

For reference the SIM_dyncomp S_define is included here. It relies extensively on the standard JEOD S-modules:

1.3.1 Preamble

```
//=========================TRICK HEADER====================
// PURPOSE:
//=================================================================================
// This simulation provides a reference implementation that can be used for
// comparison to other orbital dynamics implementations.  It is intended to
// provide a reference set for dynamics comparison.
//
// This simulation models a single vehicle in orbit around the Earth.  There
// are many adjustable configuration parameters that can be used to test out
// a suite of test cases with specific behavior.  These test cases form the
// basis for simulation to simulation comparison.
//
//          sys - Trick runtime executive and data recording routines
//         time - Representations of different clocks used in the sim
//     dynamics - Orbital dynamics
//          env - Environment: ephemeris, gravity
//          sun - Sun planetary model
//         moon - Moon planetary model
//        earth - Earth planetary model
//       vehicle - Space vehicle dynamics model
//
//=================================================================================
```

1.3.2 Configuration

- Define the job-calling intervals

```
// Define job calling intervals
#define LOW_RATE_ENV  60.00    // Low-rate environment update interval
#define DYNAMICS       0.03125 // Vehicle and plantary dynamics interval (32Hz)
```

- Include the standard job-priority definitions

```
// Define the phase initialization priorities.
#include "default_priority_settings.sm"
```

- Bring in the trick-system and jeod-system top-level modules

```
 // Include the default system classes:
#include "sim_objects/default_trick_sys.sm"
#include "jeod_sys.sm"
```

1.3.3 Time

Use the standard jeod_time.sm sim-module; in addition to the default TAI clock, add UT1, UTC, TT, and GMST. Also specify that the clocks should have their calendar representations computed out.

```
// Set up desired time types and include the JEOD time S_module
#define TIME_MODEL_UT1
#define TIME_MODEL_UTC
#define TIME_MODEL_TT
#define TIME_MODEL_GMST
#define TIME_CALENDAR_UPDATE_INTERVAL  DYNAMICS
#include "jeod_time.sm"
```

1.3.4 Dynamics

Use the standard dynamics sim-object.

```
#include "dynamics.sm"
```

1.3.5 Environment

Use the standard environment sim-object, with sun, moon and earth planetary bodies.

```
#include "environment.sm"
#include "sun_basic.sm"
#include "moon_basic.sm"
```

For earth, use the full earth model with a spherical harmonic gravity, MET atmosphere, and RNP. For legacy reasons, we use the (older) GEMT1 gravity model rather than the default GGM02C model.

```
#define JEOD_OVERRIDE_GGM02C_WITH_GEMT1
#include "earth_GGM02C_MET_RNP.sm"
```

1.3.6 Vehicle

We want some specific capabilities for our vehicle, so we will build a new vehicle sim-object based off one of the standard S-modules.

Model Headers

```
/*****************************************************************************
Vehicle Sim Object
Purpose:(Provides the vehicle object)
*****************************************************************************/
// Include headers for classes that this class contains:
##include "dynamics/body_action/include/dyn_body_init_lvlh_rot_state.hh"
##include "dynamics/derived_state/include/euler_derived_state.hh"
##include "dynamics/derived_state/include/lvlh_derived_state.hh"
##include "dynamics/derived_state/include/orb_elem_derived_state.hh"
##include "dynamics/dyn_body/include/force.hh"
##include "dynamics/dyn_body/include/torque.hh"
##include "interactions/gravity_torque/include/gravity_torque.hh"
##include "environment/atmosphere/include/atmosphere.hh"
##include "interactions/aerodynamics/include/aero_drag.hh"

// Include default data classes:
##include "interactions/aerodynamics/data/include/aero_model.hh"
```

Inheritance

Note - we use the sim-object version found in Base. This is just the definition, without the instantiation.

```
// include the base class defintion.
#include "Base/vehicle_atmosphere.sm"
class VehicleSimObject : public VehicleAtmSimObject
{
```

Content

Note - this is the content in addition to the content of the VehicleAtmSimObject.

```
 public:
  DynBodyInitLvlhRotState lvlh_init;
  EulerDerivedState     euler;
  LvlhDerivedState      lvlh;
  EulerDerivedState     lvlh_euler;
  OrbElemDerivedState   orb_elem;
  Force  force_extern;
  Torque torque_extern;
  GravityTorque grav_torque;
  SphericalHarmonicsGravityControls  earth_grav_control;
  SphericalHarmonicsGravityControls  moon_grav_control;
```

```
SphericalHarmonicsGravityControls  sun_grav_control;
AerodynamicDrag  aero_drag;

// Instances for default data:
AerodynamicDrag_aero_model_default_data    aero_drag_default_data;

//Constructor
VehicleSimObject( DynManager    & dyn_manager_,
                  METAtmosphere & atmos_,
                  WindVelocity  & wind_)
   :
   VehicleAtmSimObject( dyn_manager_, atmos_, wind_)

{
  //
  //Default data jobs
  //
  ("default_data") aero_drag_default_data.initialize ( &aero_drag );

  //
  // Initialization jobs
  //
  P_DYN  ("initialization") euler.initialize( dyn_body,
                                              dyn_manager );
  P_DYN  ("initialization") lvlh.initialize( dyn_body,
                                             dyn_manager );
  P_DYN  ("initialization") lvlh_euler.initialize( lvlh.lvlh_frame,
                                                   dyn_body,
                                                   dyn_manager );
  P_DYN  ("initialization") orb_elem.initialize( dyn_body,
                                                 dyn_manager );
  ("initialization") euler.update( );
  ("initialization") pfix.update( );
  ("initialization") lvlh.update( );
  ("initialization") lvlh_euler.update( );
  ("initialization") orb_elem.update( );
  ("initialization") grav_torque.initialize( dyn_body );
  //
  // Environment class jobs
  //
  (DYNAMICS, "environment") euler.update( );
  (DYNAMICS, "environment") lvlh.update( );
  (DYNAMICS, "environment") lvlh_euler.update( );
  (DYNAMICS, "environment") orb_elem.update( );
  //
  // Derivative class jobs
```

```
    //
    P_BODY   ("derivative") aero_drag.aero_drag(
          dyn_body.composite_body.state.trans.velocity,
          &atmos_state,
          dyn_body.structure.state.rot.T_parent_this,
          dyn_body.composite_properties.mass,
          dyn_body.composite_properties.position);
    P_PGRAV   ("derivative") grav_torque.update();
  }

 private:
  VehicleSimObject (const VehicleSimObject&);
  VehicleSimObject & operator = (const VehicleSimObject&);

};
```

Instantiation

```
VehicleSimObject vehicle ( dynamics.dyn_manager,
                           earth.atmos,
                           earth.wind_velocity);
```

1.3.7 Collect statements

Collect forces and torques

```
vcollect vehicle.dyn_body.collect.collect_effector_forc CollectForce::create {
   vehicle.force_extern
};
vcollect vehicle.dyn_body.collect.collect_environ_forc CollectForce::create {
};
vcollect vehicle.dyn_body.collect.collect_no_xmit_forc CollectForce::create {
   vehicle.aero_drag.aero_force
};
vcollect vehicle.dyn_body.collect.collect_effector_torq CollectTorque::create {
   vehicle.torque_extern
};
vcollect vehicle.dyn_body.collect.collect_environ_torq CollectTorque::create {
};
vcollect vehicle.dyn_body.collect.collect_no_xmit_torq CollectTorque::create {
   vehicle.aero_drag.aero_torque,
   vehicle.grav_torque.torque
};
```

1.3.8 Integration Loop

```
IntegLoop sim_integ_loop  (DYNAMICS) dynamics, earth, vehicle;
```

Bibliography

[1] Edwin Z. Crues, Albert A Jackson, and Jeffery C. Morris. A Process for Comparing Dynamics of Distributed Space Systems Simulations. San Diego, CA, 2009. NASA Johnson Space Center, 2009 SPRING SIMULATION INTEROPERABILITY WORKSHOP.