JSC-61777-models/dynamics/rel_kin

# JSC Engineering Orbital Dynamics Relative Kinematics Model

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

## Package Release JEOD v5.1

## Document Revision 1.0
## July 2023


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

JSC-61777-models/dynamics/rel_kin

# JSC Engineering Orbital Dynamics
# Relative Kinematics Model

**Document Revision 1.0**
**July 2023**

**Jeff Morris**

**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**

The JEOD Relative Kinematics Model enables centralized management and calculation of the states of multiple points of interest associated with a particular Dynamics Body, relative to some other reference frame. These points can be anywhere in relation to the Dynamics Body, and the other reference frame can be any type of reference frame. All points of interest can be updated in batch fashion, or individual points can be updated. The Relative Kinematics Model does no calculations on its own, but instead serves as a "manager" class that orchestrates a set of user-defined Relative Derived States as desired by the user to accomplish these capabilities.

This document describes the implementation of the Relative Kinematics Model including the model requirements, specifications, and architecture. A user guide is provided to assist with implementing the model in Trick simulations. Finally, the methods and results of verification and validation of the model are included.

# Contents

# Chapter 1

# Introduction

## 1.1 Purpose and Objectives of the Relative Kinematics Model

The purpose of the Relative Kinematics Model is to provide the JEOD v5.1 with a centralized management interface for the calculation of relative states of multiple points of interest associated with a particular Dynamics Body, relative to any desired reference frame. These points can be anywhere on, or anywhere in relation to, the Dynamics Body, and the desired reference frame can be any coordinate frame. All points of interest can be updated in batch fashion, or individual points can be updated one at a time. The Relative Kinematics Model does no calculations on its own, but instead serves as a "manager" class that orchestrates a set of user-defined Relative Derived States to accomplish these capabilities.

## 1.2 Context within JEOD

The following document is parent to this document:

- *JSC Engineering Orbital Dynamics* [4]

The Relative Kinematics Model forms a component of the dynamics suite of models within JEOD v5.1. It is located at models/dynamics/rel_kin.

## 1.3 Document History

| Author | Date | Revision | Description |
|---|---|---|---|
| Jeff Morris | April 2010 | 1.0 | Initial Version |

The Relative Kinematics Model is a model that is heavily revised for JEOD v2.0.x, though it is partly based on the Relative Kinematics Computations model released as part of JEOD v1.5.x. Thus, this document derives lightly from that model's documentation, entitled JSC Engineering

Orbital Dynamics Relative Kinematics Computations Module, most recently released with JEOD v1.5.2.

## 1.4 Document Organization

This document is formatted in accordance with the NASA Software Engineering Requirements Standard [5] and is organized into the following chapters:

**Chapter 1: Introduction** - This introduction contains four sections: purpose and objective, context within JEOD, document history, and document organization.

**Chapter 2: Product Requirements** - Describes requirements for the Relative Kinematics Model.

**Chapter 3: Product Specification** - Describes the underlying theory, architecture, and design of the Relative Kinematics Model in detail. It is organized into four sections: Conceptual Design, Mathematical Formulations, Detailed Design, and Version Inventory.

**Chapter 4: User's Guide** - Describes how to use the Relative Kinematics Model in a Trick simulation. It is broken into three sections to represent the JEOD defined user types: Analysts or users of simulations (Analysis), Integrators or developers of simulations (Integration), and Model Extenders (Extension).

**Chapter 5: Verification and Validation** - Contains verification and validation procedures and results for the Relative Kinematics Model.

# Chapter 2

# Product Requirements

This chapter identifies the requirements for the Relative Kinematics Model.

*Requirement relkin_1: Top-level Requirement*

**Requirement:**
This model shall meet the JEOD project requirements specified in the JEOD v5.1 top-level document.

**Rationale:**
This model shall, at a minimum, meet all external and internal requirements applied to the JEOD v5.1 release.

**Verification:**
Inspection

## 2.1 Data Requirements

This section identifies requirements on the data represented by the Relative Kinematics Model. These as-built requirements are based on the Relative Kinematics Model data definition header file.

*Requirement relkin_2: Relative Kinematics Data Encapsulation*

**Requirement:**
The Relative Kinematics Model shall encapsulate the following data to enable its proper functionality:

*2.1 Number of Relative States* The number of Relative Derived States being managed by the Relative Kinematics Model.

*2.2 List of Relative States* A list of pointers to the Relative Derived States being managed.

**Rationale:**

The primary purpose of the Relative Kinematics Model is to serve as a consolidated management interface for a collection of Relative Derived States; the model needs these pieces of information to properly manage the collection. This requirement constrains the design of the module data.

**Verification:**

Inspection

## 2.2 Functional Requirements

This section identifies requirements on the functional capabilities provided by the Relative Kinematics Model. These as-built requirements are based on the Relative Kinematics Model source files.

*Requirement relkin_3: Add New Relative State To List*

**Requirement:**

The Relative Kinematics Model shall perform the following actions when commanded to add a new Relative Derived State to its list of managed ones:

*3.1 Check For Duplicate Relative State* Search the list of currently managed Relative Derived States to ensure the candidate is not already being managed.

*3.2 Add Unique Relative State To List* If the candidate is found to be unique, then it is added to the list of currently managed Relative Derived States.

**Rationale:**

The ability to add new Relative Derived States to the Relative Kinematics Model is necessary for it to fulfill its purpose as a centralized management interface for Relative Derived States.

**Verification:**

Inspection, Test

*Requirement relkin_4: Remove Relative State From List*

**Requirement:**

The Relative Kinematics Model shall perform the following actions when commanded to remove a Relative Derived State from its list of managed ones:

*4.1 Check For Relative State* Search the list of currently managed Relative Derived States to ensure the candidate is being managed.

*4.2 Remove Relative State From List* If the candidate is found, then it is removed from the list of currently managed Relative Derived States.

**Rationale:**

The ability to remove Relative Derived States from the Relative Kinematics Model is necessary for it to fulfill its purpose as a centralized management interface for Relative Derived States.

**Verification:**

Inspection, Test

*Requirement relkin_5: Find Relative State In List*

**Requirement:**

The Relative Kinematics Model shall find a specified Relative Derived State in its list of managed ones when commanded to do so.

**Rationale:**

This capability is necessary both for Relative Derived State list maintenance and state updates, the two primary functions of the model.

**Verification:**

Inspection, Test

*Requirement relkin_6: Update Single Relative State*

**Requirement:**

The Relative Kinematics Model shall direct the state update of any single Derived Relative State in its list of managed ones when commanded to do so.

**Rationale:**

This is one of the core intended capabilities of the model.

**Verification:**

Inspection, Test

*Requirement relkin_7: Update All Relative States*

**Requirement:**

The Relative Kinematics Model shall direct the update of all of the Derived Relative States in its list of managed ones when commanded to do so.

**Rationale:**

This is one of the core intended capabilities of the model.

**Verification:**

Inspection, Test

# Chapter 3

# Product Specification

This chapter defines the conceptual design, the mathematical formulations, and the detailed design for the Relative Kinematics Model. It also contains the version inventory for this release of the Relative Kinematics Model.

## 3.1 Conceptual Design

The Relative Kinematics Model is designed to be the centralized management interface for groups of Relative Derived States for the JEOD v5.1. These relative state instances should all be associated with the same Dynamics Body, but can be tied to any point of interest on, in, or around that body. The relative states can also be calculated relative to any desired reference frame. All of the relative states can be updated in batch fashion, or they can be updated one at a time, or some combination of the two. Note that the Relative Kinematics Model does no calculations on its own, but instead orchestrates the set of Relative Derived States it is currently managing to accomplish these capabilities.

## 3.2 Mathematical Formulations

The Relative Kinematics Model performs no calculations on its own; it instead orchestrates instances of Relative Derived States upon command. Thus, the reader should refer to the Relative Reference Frame State-representation section of the JEOD v5.1 Derived State Model documentation [6] for information about the calculations performed via this model.

## 3.3 Detailed Design

The functionality of the Relative Kinematics Model is fully contained within a single class. This section describes the class in detail.

**Relative Kinematics Class Design**

The *relative_kinematics.hh* file contains the lone Relative Kinematics Model class, which is named "RelativeKinematics". RelativeKinematics contains the following data members that define the list of relative states currently under management by the Relative Kinematics Model:

- relative_states: A Standard Template Library (STL) list of pointers to RelativeDerivedState objects, which are the links to each of the relative states currently being managed, and
- num_rel_states: The integer number of relative states currently being managed.

In addition to the default constructor and the destructor, the RelativeKinematics class also contains the following member functions.

**add_relstate** This member function adds a given unique RelativeDerivedState to the list of ones currently being managed by the Relative Kinematics Model.

- Return: void - no returned value.
- In: RelativeDerivedState & relstate - address of the RelativeDerivedState to be added to the Relative Kinematics Model's list.

**remove_relstate** This member function removes a given RelativeDerivedState from the list of ones currently being managed by the Relative Kinematics Model.

- Return: void - no returned value.
- In: RelativeDerivedState & relstate - address of the RelativeDerivedState to be removed from the Relative Kinematics Model's list.

**find_relstate** This member function finds the RelativeDerivedState corresponding to the name provided in the Relative Kinematics Model's list.

- Return: RelativeDerivedState * - a pointer to the RelativeDerivedState found; has value "NULL" if none were found.
- In: char * relstate_name - name of the RelativeDerivedState to be found in the Relative Kinematics Model's list.

**update_single** This member function orchestrates the update of the single RelativeDerivedState provided.

- Return: void - no returned value.
- In: char * relstate_name - name of the single RelativeDerivedState to be updated.

**update_all** This member function orchestrates the update of all of the RelativeDerivedState currently in the Relative Kinematics Model's list.

- Return: void - no returned value
- Void: function takes no inputs.

Further information about the design of this model can be found in the *Reference Manual* [1].

## 3.4 Version Inventory

The following enumerates all files that are contained in this version of the Relative Kinematics Model.

```
./docs:
refman.pdf
rel_kin.pdf

./docs/tex:
makefile
rel_kinAbstract.tex
rel_kin.bib
rel_kinChapters.tex
rel_kin.sty
rel_kin.tex

./include:
relative_kinematics.hh

./src:
relative_kinematics.cc

./verif/SIM_RELKIN_VERIF:
S_define
S_overrides.mk

./verif/SIM_RELKIN_VERIF/Log_data:
log_relkin_verif.d

./verif/SIM_RELKIN_VERIF/SET_test/RUN_relkin_test:
input

./verif/SIM_RELKIN_VERIF/SET_test_val/RUN_relkin_test:
log_relkin_verif.trk
```

# Chapter 4

# User Guide

This chapter discusses how to use the Relative Kinematics Model in a Trick simulation. Usage is treated at three levels of detail: Analysis, Integration, and Extension, each targeted at one of the three main anticipated categories of JEOD v5.1 users.

The Analysis section of the user guide is intended primarily for users of pre-existing simulations. It contains an overview of a typical S_define sim object that implements the Relative Kinematics Model, but does not discuss how to edit it; the Analysis section also describes how to modify Relative Kinematics Model variables after the simulation has compiled, such as via the input file. Since the Relative Kinematics Model is primarily a manager model, no discussion of variable logging will be given, as the variables likely to be logged are members of the contained relative states and thus are covered in other JEOD v5.1 model documentation.

The Integration section of the user guide is intended for simulation developers. It describes the necessary configuration of the Relative Kinematics Model within an S_define file, and the creation of standard run directories. The Integration section assumes a thorough understanding of the preceding Analysis section of the user guide. Where applicable, the user may be directed to selected portions of the Product Specification (Chapter 3).

The Extension section of the user guide is intended primarily for developers needing to extend the capability of the Relative Kinematics Model. Such users should have a thorough understanding of how the model is used in the preceding Integration section, and of the model specification (described in Chapter 3).

Note that the Relative Kinematics Model depends completely on the Relative Derived States model, and any simulation involving implementation of the Relative Kinematics Model will thus require the successful setup of one or more RelativeDerivedState objects ([6]) as well. This model dependency will be discussed as appropriate in the following sections.

## 4.1 Analysis

The Analysis and the Integration sections will assume, for the purposes of illustration, S_define objects of the following form:

```
sim_object {

   environment/time:          TimeManager    time_manager;

} time;


sim_object {

   dynamics/dyn_manager:      DynManager     dyn_manager;

} mngr;


sim_object {

   dynamics/dyn_body:         Simple6DofDynBody dyn_body;

} veh;


sim_object {

   dynamics/rel_kin:          RelativeKinematics         relkin;
   dynamics/derived_state:    RelativeDerivedState       cm_relstate;
   dynamics/derived_state:    RelativeDerivedState       sensor1_relstate;
   dynamics/derived_state:    RelativeDerivedState       sensor2_relstate;


   P_DYN (initialization) dynamics/rel_kin:
   relkin.relkin.add_relstate (
      In RelativeDerivedState & new_relstate  = relkin.cm_relstate);

   P_DYN (initialization) dynamics/rel_kin:
   relkin.relkin.add_relstate (
      In RelativeDerivedState & new_relstate  = relkin.sensor1_relstate);

   P_DYN (initialization) dynamics/rel_kin:
   relkin.relkin.add_relstate (
      In RelativeDerivedState & new_relstate  = relkin.sensor2_relstate);


   (DYNAMICS, environment) dynamics/rel_kin:
   relkin.relkin.update_all ( );

   P_ENV Imngr (derivative) dynamics/rel_kin:
```

```
   relkin.relkin.update_single (
      In char                 * relstate_name = relkin.cm_relstate.name);

} relkin;
```

Note that this code is only representative of sim objects necessary for this discussion, and does not hold a complete implementation. For full implementation details on the Time model, please see the JEOD v5.1 Time Representations Model documentation [7]; for full implementation details on the Dynamics Manager model, please see the JEOD v5.1 Dynamics Manager Model documentation [2]; for full implementation details on the Simple Six-DOF Dynamics Body model, please see the appropriate section of the JEOD v5.1 Dynamics Body Model documentation [3]; and for full implementation details on the Relative Derived States model, please see the Relative Reference Frame State-representation section of the JEOD v5.1 Derived State Model documentation [6].

The input files for the Relative Kinematics Model are trivial. There are simply no data fields unique to the Relative Kinematics Model beyond those in the RelativeDerivedState instances it contains, other than an integer count of the number of currently contained RelativeDerivedStates. Since this value is automatically incremented when RelativeDerivedState instances are added to the Relative Kinematics Model's list, there are no unique settings required in input files for this model. (Note: please consult the appropriate section of [6] for assistance in implementing the Relative Derived States model, including its input file settings.) Instead, all capabilities of this model are exercised by declarations in the simulation's S_define rather than the manipulation of settings via input files or similar.

## 4.2 Integration

This section describes the process of implementing the Relative Kinematics Model in a Trick simulation, and will use the same example S_define found in the Analysis section for illustration. Please again note that this code is only representative of sim objects necessary for this discussion, and does not describe a complete implementation. For full implementation details on the Time model, please see the JEOD v5.1 Time Representations Model documentation [7]. For full implementation details on the Dynamics Manager model, please see the JEOD v5.1 Dynamics Manager Model documentation [2]. For full implementation details on the Simple Six-DOF Dynamics Body model, please see the appropriate section of the JEOD v5.1 Dynamics Body Model documentation [3]. For full implementation details on the Relative Derived States model, please see the Relative Reference Frame State-representation section of the JEOD v5.1 Derived State Model documentation [6].

To successfully integrate the Relative Kinematics Model into a Trick simulation, the S_define must contain instantiations of the following objects:

- A TimeManager object,
- A DynManager object,
- A DynBody object or one of its derived classes such as Simple6DofDynBody,
- One or more RelativeDerivedState objects for the Relative Kinematics Model to manage, and

- A RelativeKinematics object.

As noted above, the TimeManager, DynManager, DynBody or Simple6DofDynBody, and RelativeDerivedState classes must be correctly initialized and set up for successful simulation operation, though discussion of such setup for those classes is outside of the scope of this document; documentation specific to each of the applicable models should be consulted for those details.

The main object for the Relative Kinematics Model is the RelativeKinematics class, which is instantiated in the example code above via the line:

```
dynamics/rel_kin:          RelativeKinematics         relkin;
```

After setting up the instantiation of a RelativeKinematics object, the next step is to add the RelativeDerivedStates that it is to manage to its list. (Note that they should be properly initialized first, which is outside the scope of this document; please consult the model-specific documentation for assistance in implementing this.) This is accomplished by invoking the Relative Kinematics Model *add_relstate* member function as an initialization class job, as shown in the example above and reproduced here:

```
P_DYN (initialization) dynamics/rel_kin:
relkin.relkin.add_relstate (
   In RelativeDerivedState & new_relstate  = relkin.cm_relstate);

P_DYN (initialization) dynamics/rel_kin:
relkin.relkin.add_relstate (
   In RelativeDerivedState & new_relstate  = relkin.sensor1_relstate);

P_DYN (initialization) dynamics/rel_kin:
relkin.relkin.add_relstate (
   In RelativeDerivedState & new_relstate  = relkin.sensor2_relstate);
```

This example shows the addition of three separate relative states to the Relative Kinematics Model's list, each of which must be added individually via a separate call to the *add_relstate* function.

The final step required to integrate the Relative Kinematics Model into a Trick simulation is to command the model to update the relative states at the desired time invervals. This is done by invoking either the Relative Kinematics Model *update_single* or *update_all* member functions as desired:

```
   (DYNAMICS, environment) dynamics/rel_kin:
   relkin.relkin.update_all ( );

   P_ENV Imngr (derivative) dynamics/rel_kin:
   relkin.relkin.update_single (
      In char                 * relstate_name = relkin.cm_relstate.name);
```

In this example code snippet, all of the currently managed relative states are being updated in batch fashion as an environment class job at the "DYNAMICS" rate (the definition of which is outside the scope of this document), while the relative state "cm_relstate" is additionally being updated by itself as a derivative class job and thus at the so-called derivative rate. Note that either function can be called at any frequency desired; the choices here are shown merely for introduction to the model's capabilities.

No further steps are required for integration of the Relative Kinematics Model into a Trick simulation. As its purpose is primarily to serve as an interface by which the user can more easily manage multiple RelativeDerivedState objects all at once, the Relative Kinematics Model itself performs no further tasks beyond the ones just illustrated.

## 4.3 Extension

Because the Relative Kinematics Model was implemented in a very specific way to perform a very specific interface management function, it is not intended to be extensible.

# Chapter 5

# Verification and Validation

## 5.1 Verification

*Inspection relkin_1: Top-level Inspection*

This document structure, the code, and associated files have been inspected, and together completely satisfy requirement relkin_1.

*Inspection relkin_2: Data Requirements Inspection*

By inspection, the data structures of the Relative Kinematics Model completely satisfy requirement relkin_2.

*Inspection relkin_3: Functional Requirements Inspection*

By inspection, the as-written function code of the Relative Kinematics Model satisfies requirements relkin_3, relkin_5, relkin_6, and relkin_7.

## 5.2 Validation

For the single test case, a Trick simulation was run with an appropriately configured input file. This run can be found in its own directory in the SET_test sub-directory of SIM_RELKIN_VERIF.

*Test relkin_1: Relative Kinematics Model Test*

**Purpose:**

This test case is designed to examine the ability of the Relative Kinematics Model to add new relative states to its internal list, to find relative states that are already in its internal list, to update individual relative states when so commanded, and to update all currently managed

relative states when commanded. In doing so, the test case examines all functional aspects of the Relative Kinematics Model.

**Run directory:**
RUN_relkin_test

**Requirements:**
By passing this test, this model satisfies requirements relkin_3, relkin_5, relkin_6, and relkin_7.

**Procedure:**
Since the model performs no calculations of its own, but instead merely directs instances of the RelativeDerivedState class to perform their calculations as appropriate, simple successful execution of a simulation containing both a properly implemented instance of the Relative Kinematics Model, and properly implemented instances of the RelativeDerivedState class, is sufficient to fully demonstrate the capabilities of the model under examination.

**Results:**
The simulation completed the run successfully. Since calls to all of the model's member functions are included either directly or indirectly in the test case (note that function *find_relstate* is exercised by both the *add_relstate* and *update_single* even though it is not explicitly called in the S_define), requirements relkin_3, relkin_5, relkin_6, and relkin_7 are demonstrated to be satisfied by this test.

## 5.3 Requirements Traceability

Table 5.1: Requirements Traceability

| **Requirement** | **Inspection and Testing** |
|---|---|
| relkin_1 - Top-level Requirement | Insp. relkin_1 |
| relkin_2 - Relative Kinematics Data Encapsulation | Insp. relkin_2 |
| relkin_3 - Add New Relative State To List | Insp. relkin_3<br>Test relkin_1 |
| relkin_5 - Find Relative State In List | Insp. relkin_3<br>Test relkin_1 |
| relkin_6 - Update Single Relative State | Insp. relkin_3<br>Test relkin_1 |
| relkin_7 - Update All Relative States | Insp. relkin_3<br>Test relkin_1 |

# Bibliography


[1] Generated by doxygen. *Relative Kinematics Model Reference Manual*. National Aeronautics and Space Administration, Johnson Space Center, Automation, Robotics & Simulation Division, Simulation and Graphics Branch, 2101 NASA Parkway, Houston, Texas, 77058, July 2023.

[2] Hammen, D. Dynamics Manager Model. Technical Report JSC-61777-dynamics/dyn_manager, NASA, Johnson Space Center, Houston, Texas, July 2023.

[3] Hammen, D. Dynamic Body Model. Technical Report JSC-61777-dynamics/dyn_body, NASA, Johnson Space Center, Houston, Texas, July 2023.

[4] Jackson, A., Thebeau, C. JSC Engineering Orbital Dynamics. Technical Report JSC-61777-docs, NASA, Johnson Space Center, Houston, Texas, July 2023.

[5] NASA. NASA Software Engineering Requirements. Technical Report NPR-7150.2, NASA, NASA Headquarters, Washington, D.C., September 2004.

[6] Turner, G. Derived State Model. Technical Report JSC-61777-dynamics/derived_state, NASA, Johnson Space Center, Houston, Texas, July 2023.

[7] Turner, G. Time Model. Technical Report JSC-61777-environment/time, NASA, Johnson Space Center, Houston, Texas, July 2023.