JSC-61777-models/utils/ref_frames

JSC Engineering Orbital Dynamics Reference Frame Model

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

Package Release JEOD v5.0

Document Revision 1.1
July 2022

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

JSC Engineering Orbital Dynamics Reference Frame Model

Document Revision 1.1
July 2022

Andrew Spencer

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

For a vehicle in orbit, there are many interrelated reference frames that can be defined, often with respect to one another. Inertial frames, planet fixed frames, vehicle body frames, and vehicle structural frames are all important concepts when modeling a vehicle in planetary orbit. Additionally, knowledge of the state of the vehicle, often with respect to many interrelated reference frames, is critical to successful modeling of the vehicle dynamics.

The JEOD Reference Frame Model provides a structure for specifying trees of reference frames, as well as the relative states between them. Additionally, this model contains the tools for iterating over a generic tree structure, which allows for implementation of common tree algorithms over other models.

Additionally, the Reference Frame Model also provides a Reference Frame Manager class, used to perform common Reference Frame related tasks, such as registering and searching for reference frames, building a reference frame tree, and interacting with the reference frame subscription functionality.

Contents

1 Introduction **1**

1.1 Model Description 1

1.2 Document History 1

1.3 Document Organization 2

2 Product Requirements **3**

3 Product Specification **6**

3.1 Conceptual Design 6

3.1.1 RefFrameOwner 6

3.1.2 RefFrameItems 6

3.1.3 RefFrameMessages 6

3.1.4 RefFrameTrans 6

3.1.5 RefFrameRot 7

3.1.6 RefFrameState 7

3.1.7 RefFrame 8

3.1.8 RefFrameLinks 8

3.1.9 TreeLinksIterator 8

3.1.10 RefFrameManager 8

3.2 Mathematical Formulations 8

3.2.1 Nomenclature 9

3.2.2 Operators 9

3.3 Detailed Design 12

3.4 Inventory 12

4 User Guide **15**

4.1 Analysis . . . 15
4.2 Integration . . . 17
4.2.1 Reference Frame Utilities . . . 18
4.2.2 Building a Rooted Tree of Reference Frames . . . 19
4.2.3 Calculating Relative States . . . 21
4.2.4 Using the Reference Frame Manager . . . 21
4.3 Extension . . . 24
4.3.1 Creating a Reference Frame Owner . . . 24
4.3.2 Extending the TreeLinksIterator Template Usage . . . 24
4.3.3 Extending the RefFrameManager Class . . . 27

5 Verification and Validation **28**
5.1 Verification . . . 28
5.2 Validation . . . 33
5.3 Metrics . . . 54
5.3.1 Code Metrics . . . 54
5.4 Requirements Traceability . . . 65

Chapter 1

Introduction

1.1 Model Description

In dynamic applications, reference frames are inherently a key concept. They are especially crucial in orbital dynamic applications, where reference frames and in-depth knowledge of relative kinematics are commonly used. Examples of these applications include calculating relative kinematics between vehicles in proximity operations, calculation of vehicle state in the LVLH or North-East-Down frames [12], or converting a vehicle position from a planet inertial frame to one that is planet fixed. Additionally, many points of interest on a vehicle can be represented with reference frames, often defined relative to a central reference frame about which the vehicle is being integrated.

The Reference Frame Model gives tools to the user to create a system of reference frames, contained in a rooted tree structure. Each reference frame can have its full state (translational and rotational) specified with respect to its parent frame in the tree, and within that tree the relative state of any reference frame can be calculated with respect to any other frame contained in that tree. Additionally, a system for iterating over generic rooted tree structures is available, which aids in the implementation of other tree systems.

Additionally, the Reference Frame Model supplies a utility for managing reference frames. This utility, called the Reference Frame Manager, provides a searchable registry for reference frames, as well as common functionalities for reference frame interaction. These functionalities include building and re-building reference frame trees and accessing the subscription functionality of the reference frames.

1.2 Document History

Author	Date	Revision	Description
Andrew Spencer	November, 2009	1.0	Initial Version
Andrew Spencer	October, 2010	1.1	Add Reference Frame Manager, metrics

The following document is parent to this document:

- *JSC Engineering Orbital Dynamics* [6]

1.3 Document Organization

This document is formatted in accordance with the NASA Software Engineering Requirements Standard [7] and is organized into the following chapters:

Chapter 1: Introduction - This introduction contains three sections: description of model, document history, and organization. The first section provides the introduction to the Reference Frame Model and its reason for existence. It also contains a brief description of the interconnections with other models, and references to any supporting documents. The second section displays the history of this document which includes author, date, and reason for each revision; it also lists the document that is parent to this one. The final section contains a description of the how the document is organized.

Chapter 2: Product Requirements - Describes requirements for the Reference Frame Model.

Chapter 3: Product Specification - Describes the underlying theory, architecture, and design of the Reference Frame Model in detail. It is organized in three sections: Conceptual Design, Mathematical Formulations, and Detailed Design.

Chapter 4: User Guide - Describes how to use the Reference Frame 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 Reference Frame Model verification and validation procedures and results.

Chapter 2

Product Requirements

This chapter will describe the requirements for the Reference Frame Model.

Requirement refframes_1: Top-level requirement

Requirement:
This model shall meet the JEOD project requirements specified in the JEOD v5.0 top-level document.

Rationale:
This model shall, at a minimum, meet all external and internal requirements applied to the JEOD v5.0 release.

Verification:
Inspection

Requirement refframes_2: Reference Frame Representation Requirement

Requirement:
The Reference Frame Model shall describe a frame of reference characterized by an origin and a set of three orthogonal axes.

Rationale:
This representation of a reference frame, of three orthogonal axes in Cartesian three space, is the most recognized and most intuitive representation of free space.

Verification:
The verification for this item shall be done by inspection.

Requirement refframes_3: Reference Frame State Requirement

Requirement:
The Reference Frame Model shall provide a mechanism for specifying the translational and

rotational states of an object in space (specifically, Cartesian three space).

Rationale:

Providing information about translational and rotational states is a basic requirement for JEOD, and is a basic requirement for any reference frame system.

Verification:

The verification for this requirement shall be done by inspection.

Requirement refframes_4: Reference Frame Tree Structure

Requirement:

The Reference Frame Model shall provide reference frames that are nodes within a rooted tree structure.

Rationale:

Placing the reference frames in a rooted tree structure allows for concise, well understood relationships between frames to be defined and utilized.

Verification:

The verification for this requirement shall be done by inspection.

Requirement refframes_5: Reference Frame Relative State Calculations

Requirement:

The Reference Frame Model shall provide functionality to determine the relative state between any two reference frames found within the same rooted tree.

Rationale:

Knowing relative state is extremely important functionality when dealing with simulations based on rendezvous and proximity operations.

Verification:

The verification for this requirement shall be done by testing.

Requirement refframes_6: Reference Frame Active/Inactive Functionality

Requirement:

The Reference Frame Model shall provide functionality to activate or make inactive any reference frame.

Rationale:

Being able to declare a reference frame inactive when it is not in use saves on computational complexity at runtime.

Verification:

The verification for this requirement shall be done by inspection.

Requirement refframes_7: Reference Frame Subscription Functionality

Requirement:

The Reference Frame Model shall provide a subscription functionality for indicating that another entity requires the reference frame to be active.

Rationale:

The Reference Frame Model is intended as a utility model, to be used by many other models. Indicating the necessity of a specific reference frame by another model is vital to the JEOD architecture.

Verification:

The verification for this requirement shall be done by inspection.

Requirement refframes_8: Reference Frame Manager

Requirement:

The Reference Frame Model shall provide a Reference Frame Manager utility.

8.1 Searchable Registry The Reference Frame Manager utility shall provide a searchable registry of reference frames.

8.2 Tree Building Functionality The Reference Frame Manager utility shall provide the ability to build a reference frame tree (one node at a time).

8.3 Subscription Mechanism Access The Reference Frame manager utility shall provide by-name and by-reference access to the reference frame subscription mechanism to enter a subscription, revoke a subscription, and check whether a frame has subscribers.

Rationale:

Reference Frames are a vital part of all JEOD simulations, and these are all common tasks for Reference Frames.

Verification:

Inspection.

Chapter 3

Product Specification

3.1 Conceptual Design

This section will present the conceptual design for the Reference Frame Model and the generic tree iterator utilities included in this model.

3.1.1 RefFrameOwner

The RefFrameOwner class provides an interface for any class that will be the "owner" of a reference frame. This means that, if a class inherits from RefFrameOwner, it is expected to be responsible for updating the state of that reference frame. This allows for a common interface to be used for all other objects that wish to own a reference frame.

3.1.2 RefFrameItems

The RefFrameItems class supplies an enumeration that identifies major components (position, velocity, attitude, attitude rates) of a reference frame state. For example, this enumeration can be used to identify which of the major components have been set in a particular reference frame, including all combinations of the four components. Additionally, the class also contains utility functions for working with this enumeration.

3.1.3 RefFrameMessages

The RefFrameMessages class supplies messages associated with the Reference Frame Model for use with the JEOD Message Handler [8].

3.1.4 RefFrameTrans

The RefFrameTrans class contains the current translational state of a reference frame with respect to another reference frame. For illustrative purposes, the state of the frame in question will be

Frame A, and the frame that the state is with respect to will be Frame B.

The RefFrameTrans class then contains the following information:

- Position, which represents the translational position of Frame A, with respect to Frame B, expressed in the coordinates of Frame B, and
- Velocity, which represents the translational velocity of Frame A, with respect to Frame B, expressed in the coordinates of Frame B.

The class also contains utility functions, such as copying a translational state as well as initializing the translational state to zero.

3.1.5 RefFrameRot

The RefFrameRot class contains the current rotational state of a reference frame with respect to another reference frame. For illustrative purposes, the state of the frame in question will be Frame A, and the frame that the state is with respect to will be Frame B.

The RefFrameRot class then contains the following information:

- Q_parent_this, a left transformation quaternion from Frame B to Frame A,
- T_parent_this, a transformation matrix from Frame B to Frame A,
- ang_vel_this, the angular velocity of Frame A, with respect to Frame B, expressed in the coordinates of Frame A,
- ang_vel_mag, the magnitude of "ang_vel_this,",
- ang_vel_unit, the unit vector of "ang_vel_this."

Note that there is some duplication of information in this particular class, such as Q_parent_this and T_parent_this representing the same information. Functions exist to calculate all related values from the first set, as well as functions that initialize the state and copy the information in a second, user supplied RefFrameRot.

3.1.6 RefFrameState

The RefFrameState class encapsulates the rotational and translational state objects for a particular reference frame. It also provides similar functionality, such as initializing an otherwise empty state and copying one state from another.

Additionally, the RefFrameState class provides in depth mathematical functionality for determining kinematic states of one reference with respect to another. The foundation for these functions will be presented in the Mathematical Formulation section, and the interface to the user for the functions that implement this math will be presented in the Detailed Design section.

3.1.7 RefFrame

The RefFrame class represents a single reference frame in a rooted tree of reference frames. It is a named, time-stamped object that can be subscribed to as well as owned by another object. It is part of a rooted tree structure of other RefFrame objects, where it has a parent (unless it is the root node of the tree), and children RefFrames that the RefFrame in question is a parent to. A RefFrame also contains a RefFrameState object that specifies the RefFrame's state with respect to its parent frame, as described in the RefFrameTrans and RefFrameRot class descriptions.

Additionally, a RefFrame contains functionality associated with the rooted tree structure, including re-organizing the tree as well as calculating the relative position or relative state of any RefFrame with respect to any other RefFrame found in the same tree.

3.1.8 RefFrameLinks

The RefFrameLinks class is the RefFrame specific extension of the TreeLinksIterator class. It provides the rooted tree functionality used in the reference frame implementation. It encapsulates the links between reference frames in the rooted tree.

3.1.9 TreeLinksIterator

The TreeLinksIterator class is a templatized, generic class, akin to a "pure virtual template". It provides common functionality for rooted tree structures.

3.1.10 RefFrameManager

The RefFrameManager provides utility and management functions for reference frames in a JEOD-based simulation. It provides the following main functionalites:

- Tracking of reference frames,
- By name reference frame lookup,
- Access to the subcription functionalities of the reference frames.

The RefFrameManager also acts as a base class to the EphemeridesManager [9], which builds on the basic functionality presented here.

3.2 Mathematical Formulations

The mathematical formulation for the Reference Frame Model is concerned mainly with the calculation of the relative state of a reference frame with respect to another reference frame contained in the same tree. The relative states found between the reference frames, in the tree, often contain rotating elements, which complicate the effort of calculating the full set of relative kinematics

between the reference frames. This section will present a concise nomenclature, as well as basic mathematical principles, for calculating these relative kinematics.

3.2.1 Nomenclature

The nomenclature used in this section is as follows:

- $x_{A \to B:C}$ = Position of frame B origin with respect to frame A origin, expressed in frame C coordinates (observer is fixed with respect to frame C).
- $v_{A \to B:C}$ = Velocity of frame B origin with respect to frame A origin, expressed in frame C (observer is fixed with respect to frame C).
- $w_{A \to B:C}$ = Angular velocity of frame B axes with respect to frame A axes expressed in frame C (observer is fixed with respect to frame C).
- $T_{A \to B}$ = Column vector transformation matrix from frame A to frame B.

These values can also be expressed in the following short hand notation:

- $x_{A:B} = x_{A \to B:A}$ (Location of frame B origin in frame A coordinates.)
- $v_{A:B} = v_{A \to B:A}$ (Velocity of frame B origin in frame A coordinates.)
- $T_{A:B} = T_{A \to B}$ (The transformation matrix from frame A to frame B).
- $w_{A:B} = w_{A \to B:B}$ (The angular velocity of frame B, with respect to frame A, expressed in frame B. Note that $w_{A:B}$ is expressed in frame B coordinates.)
- $S_{A:B} = x_{A:B}$, $V_{A:B}$, $T_{A:B}$, $w_{A:B}$. In other words, the state of frame B with respect to frame A.

3.2.2 Operators

This section will explain and detail the four main operators associated with calculating reference frame state with respect to other reference frames. The four main operators are:

- assignment (=),
- unary negation (-),
- addition (+), and
- subtraction (-).

It should be noted that subtraction is, in reality, the combination of an addition to the negation of the subtracted term, hence sharing a symbol with the unary negation operator. The following sections will explain the exact meaning of these operators, and give their formulations.

Assignment (=)

The assignment operator for reference frame states is trivial. If the relative state of reference frame B with respect to reference frame A is found to be equivalent to the relative state of reference frame D with respect to reference frame C, then:

$$x_{A:B} = x_{C:D} \tag{3.1}$$
$$v_{A:B} = v_{C:D} \tag{3.2}$$
$$T_{A:B} = T_{C:D} \tag{3.3}$$
$$w_{A:B} = w_{C:D} \tag{3.4}$$

Unary Negation (-)

Negation of a reference frame state serves to flip which reference frame the state is with respect to, and which reference frame it is the state of. In the nomenclature defined for this section, this is equivalent to saying:

$$S_{B:A} = -S_{A:B} \tag{3.5}$$

This is to say, that the negation of the state of reference frame B with respect to A is the state of reference frame A with respect to B.

The full, piece by piece definition is then:

$$T_{B:A} = T_{A:B}^{T} \tag{3.6}$$
$$w_{B:A} = -(T_{B:A} * w_{A:B}) \tag{3.7}$$
$$x_{B:A} = -(T_{A:B} * x_{A:B}) \tag{3.8}$$
$$v_{B:A} = -(T_{A:B} * v_{A:B} + w_{A:B} X x_{B:A}) \tag{3.9}$$

Addition (+)

In terms of reference frame states, the addition operator acts to combine two states, thus given two states $S_{A:B}$ and $S_{B:C}$, the sum of their addition would be:

$$S_{A:C} = S_{A:B} + S_{B:C} \tag{3.10}$$

In other words, in terms of the addition operator being defined here, the sum of the state of reference frame B with respect to A, and the state of reference frame C with respect to B, is the state of reference frame C with respect to A.

The piece by piece, individual portions of this addition are as follows:

$$T_{A:C} = T_{B:C} * T_{A:B} \tag{3.11}$$
$$w_{A:C} = T_{B:C} * w_{A:B} + w_{B:C} \tag{3.12}$$
$$x_{A:C} = x_{A:B} + T^{T}_{A:B} * x_{B:C} \tag{3.13}$$
$$v_{A:C} = v_{A:B} + T^{T}_{A:B} * (v_{B:C} + w_{A:B} X x_{B:C}) \tag{3.14}$$

There are two very special properties of this addition operator that must be noted. First, this addition is non-commutative. Thus,

$$S_{A:B} + S_{B:C} \neq S_{B:C} + S_{A:B} \tag{3.15}$$

The second property is, that only a reference frame state with respect to a specific frame may be added to, on the right, a reference frame state whose subject is the previously mentioned specific frame. In other words, while the following addition is valid:

$$S_{A:B} + S_{B:C} \tag{3.16}$$

the following is not:

$$S_{A:B} + S_{C:D} \tag{3.17}$$

Again, this property states that, if the left hand addend in a reference frame addition operator has a subject of frame X, then the right hand addend must be with respect to this same frame X. Otherwise it is an invalid addition.

Subtraction (-)

As mentioned before, the subtraction operator for reference frames is a combination of the negation operator and the addition operator. Thus, the subtraction of two reference frames is as follows:

$$S_{C:A} = S_{C:B} - S_{A:B} = S_{C:B} + S_{B:A} \tag{3.18}$$

Similarly, the following is also true:

$$S_{C:A} = -S_{B:C} + S_{B:A} = S_{C:B} + S_{B:A} \tag{3.19}$$

Note that the results of all subtractions (in the end, an addition) must follow the same rules set out in the Addition section.

3.3 Detailed Design

The complete API for the Reference Frame Model can be found in the *Reference Manual* [1].

3.4 Inventory

All Reference Frame Model files are located in the directory
`${JEOD_HOME}/models/utils/ref_frames`. Relative to this directory,

- Header and source files are located in the model `include` and `src` subdirectories. Table 3.1 lists the configuration-managed files in these directories.
- Documentation files are located in the model `docs` subdirectory. See table 3.2 for a listing of the configuration-managed files in this directory.
- Verification files are located in the model `verif` subdirectory. See table 3.3 for a listing of the configuration-managed files in this directory.

Table 3.1: Source Files

File Name
include/base_ref_frame_manager.hh
include/class_declarations.hh
include/ref_frame.hh
include/ref_frame_inline.hh
include/ref_frame_interface.hh
include/ref_frame_items.hh
include/ref_frame_items_inline.hh
include/ref_frame_links.hh
include/ref_frame_manager.hh
include/ref_frame_messages.hh
include/ref_frame_state.hh
include/ref_frame_state_inline.hh
include/subscription.hh
include/tree_links.hh
include/tree_links_iterator.hh
src/ref_frame.cc
src/ref_frame_compute_relative_state.cc
src/ref_frame_items.cc
src/ref_frame_manager.cc
src/ref_frame_messages.cc

Continued on next page

Table 3.1: Source Files (continued from previous page)

File Name
src/ref_frame_set_name.cc
src/ref_frame_state.cc
src/subscription.cc

Table 3.2: Documentation Files

File Name
docs/ref_frames.pdf
docs/refman.pdf
docs/tex/makefile
docs/tex/model_name.mk
docs/tex/ref_frames.bib
docs/tex/ref_frames.sty
docs/tex/ref_frames.tex
docs/tex/ref_framesAbstract.tex
docs/tex/ref_framesChapters.tex
docs/tex/figs/verif_case_1.jpg
docs/tex/figs/verif_case_2.jpg

Table 3.3: Verification Files

File Name
verif/SIM_REF_FRAMES/S_define
verif/SIM_REF_FRAMES/S_overrides.mk
verif/SIM_REF_FRAMES/DP_Product/DP_validation.xml
verif/SIM_REF_FRAMES/Log_data/REF_FRAME_ver_rec.py
verif/SIM_REF_FRAMES/Modified_data/default_mods.py
verif/SIM_REF_FRAMES/SET_test/RUN_ref_frame_ver1/input.py
verif/SIM_REF_FRAMES/SET_test/RUN_ref_frame_ver2/input.py
verif/SIM_REF_FRAMES/SET_test/RUN_ref_frame_ver3/input.py
verif/SIM_REF_FRAMES/SET_test_val/RUN_ref_frame_ver1/input.py
verif/SIM_REF_FRAMES/SET_test_val/RUN_ref_frame_ver1/log_REF_FRAME_VER.header
verif/SIM_REF_FRAMES/SET_test_val/RUN_ref_frame_ver1/log_REF_FRAME_VER.trk

Continued on next page

Table 3.3: Verification Files (continued from previous page)

File Name
verif/SIM_REF_FRAMES/SET_test_val/RUN_ref_frame_ver2/input.py
verif/SIM_REF_FRAMES/SET_test_val/RUN_ref_frame_ver2/log_REF_FRAME_VER.header
verif/SIM_REF_FRAMES/SET_test_val/RUN_ref_frame_ver2/log_REF_FRAME_VER.trk
verif/SIM_REF_FRAMES/SET_test_val/RUN_ref_frame_ver3/input.py
verif/SIM_REF_FRAMES/SET_test_val/RUN_ref_frame_ver3/log_REF_FRAME_VER.header
verif/SIM_REF_FRAMES/SET_test_val/RUN_ref_frame_ver3/log_REF_FRAME_VER.trk

Chapter 4

User Guide

The Analysis section of the user guide is intended primarily for users of pre-existing simulations. It contains:

- A description of how to modify Reference Frame Model variables after the simulation has compiled, including an in-depth discussion of the input file,
- An overview of how to interpret (but not edit) the S_define file,
- A sample of some of the typical variables that may be logged.

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

The Extension section of the user guide is intended primarily for developers needing to extend the capability of the Reference Frame 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).

4.1 Analysis

This section will use the following example S_define for illustrative purposes. The Reference Frame Model needs little interaction with other models to function, and this example is extremely simple.

```
sim_object {

   RefFrame frame;

} object;
```

The Reference Frame Model is a unique model in JEOD, as it is not directly a model but is instead considered a utility, to be used in other models. Most instances of RefFrame will be used indirectly through other objects. Examples of where RefFrame objects are utilized and available to users include:

- The Dynamic Body Model [2],
- The Ephemeris Model [9],
- The Body Action Model [3],
- The Derived State Model [10].

Individual instructions for accessing RefFrames associated with these classes can be found in the respective model's documentation. This section will only cover inputs and ouputs directly from the Reference Frame Model.

To aid in search functions, a RefFrame object can have a name associated with it. This can be set through a Trick input file by a user with the following command:

```
object.frame.name = "user_set_name";
```

The main input/output from a reference frame for an Analysis type user is the current state associated with the reference frame. Inputs to these variables are most often done prior to initialization, especially when using the Body Action Model [3] to initialize dynamic bodies in the simulation. It is possible for the following parameters to be set in the reference frame state:

```
// The position of the frame, with respect to its parent, in its parent
// frame in meters
object.frame.state.trans.position[0] = 1.0, 2.0, 3.0;

// The velocity of the frame, with respect to its parent, in its parent
// frame, in meters per second
object.frame.state.trans.velocity[0] = 1.0, 2.0, 3.0;

// The left transformation quaternion from the reference frame's parent to
// this reference frame
object.frame.state.rot.Q_parent_this = input;

// The transformation matrix from the reference frame's parent to
//this reference frame
object.frame.state.rot.T_parent_this[0][0] = 1, 0, 0;
object.frame.state.rot.T_parent_this[1][0] = 0, 1, 0;
object.frame.state.rot.T_parent_this[2][0] = 0, 0, 1;

// The angular velocity of this reference frame, with respect to its parent,
// expressed in this reference frame, in radians per second
```

```
object.frame.state.rot.ang_vel_this[0] = 4.0, 0.0, 0.0;

// The magnitude of "ang_vel_this", in radians per second
object.frame.state.rot.ang_vel_mag = 4.0;

// The unit vector of "ang_vel_this"
object.frame.state.rot.ang_vel_unit = 1.0, 0.0, 0.0;
```

Note that a full demonstration of how to input values to the quaternion "Q_parent_this" is not given, and that the full details can be found in the Quaternion documentation [4]. Also, because "Q_parent_this" and "T_parent_this" are redundant values, it is not necessary to specfiy them both upon input. This code snippet only gives examples of how to access each of these variables, and only one is necessary to define the attitude of the object (which one depends on how the particular simulation is set up, see the Integration section for further details). The same is also true of the values "ang_vel_this," "ang_vel_mag" and "ang_vel_unit."

Note that a RefFrameState can also be instantiated separately from a reference frame, and be populated with values of a particular reference frame with respect to a reference frame that is not necessarily its parent. This can happen often in other models found in JEOD. When this is the case, the exact meaning of the data contained in the RefFrameState will be explained in the respective model's documentation, and the components of the state can be accessed similar to the manner shown above.

All of these values may be both entered at initialization time, as well as logged during simulation runtime. Additional parameters that can be logged during simulation runtime include:

```
// a count of the number of subscribers a reference frame has
object.frame.subscribers;

// the time stamp, in dynamic time seconds, of the last time the frame
// was updated
object.frame.update_time;
```

4.2 Integration

As mentioned in the analysis section, the Reference Frame Model is a utility model in JEOD, and is not meant to be used as a stand-alone entity. As such, a reference frame itself is not meant to be integrated into an S_define directly. Instead, it is meant to be used as a utility when building other models. Thus, this Integration section will concentrate not on integrating the Reference Frame Model into an S_define, but instead on the usage of reference frames when they are included in code for a separate model.

There are four main topics necessary when integrating reference frames into a model: using reference frames utility functions, building the rooted tree of reference frames, using the rooted tree to calculate relative states between the reference frames, and using the Reference Frame Manager as a front end to interacting with reference frames.

4.2.1 Reference Frame Utilities

The RefFrame class has a series of functions for setting the name of a reference frame by joining a set of individual names with '.'s. There are seven versions of this, all taking consecutively more C style strings as arguments. For example, the prototype that takes 4 arguments is:

```
void set_name(
   const char* name_item1,
   const char* name_item2,
   const char* name_item3,
   const char* name_item4);
```

The function will then take each consecutive argument, conjoin them together with a '.', and set that as the name of the reference frame.

Three functions are associated with if the reference frame is active or inactive. They are:

```
virtual void activate (void);

virtual void deactivate (void);

virtual bool is_active (void) const;
```

These are simple functions that set an active flag (in the form of a bool), or return it. This functionality is most often used through the JEOD Dynamics Manager [5] and the JEOD Ephemeris Model [9]. Care should be taken in understanding these models before this activation functionality is used.

Similarly, there are utility functions to subscribe to the frame. These functions will not only adjust the active flag accordingly, but will also adjust the number of subscribers of the reference frame accordingly. These functions are:

```
virtual void subscribe (void);

virtual void unsubscribe (void);

virtual unsigned int subscriptions (void) const;
```

The owner of the frame, usually interpreted as the object that contains or updates the reference frame, has similar functionality.

```
virtual void set_owner (RefFrameOwner * new_owner);

virtual RefFrameOwner * get_owner (void) const;
```

There are also functions related to the time-stamping (update of the last dyn time [11] the reference frame was updated) of the reference frame.

```
virtual void set_timestamp (double time);

virtual double timestamp (void) const;
```

Finally, when working with reference frames and specifically reference frame states, there are a set of handy functions and data members associated with the class RefFrameItems. This class gives various utilities for identifying a subset of the four major components that make up a reference frame state (position, velocity, attitude, attitude rate). An instantiated object of the RefFrameItems type can represent any combination of this set, including full and empty instantiations. This class also has a variety of utility functions for comparison of sets, adding and subtracting from sets, as well as creating a string that will identify a set in a human readable way. For full documentation of the utility functions associated with this class, please see the Detailed Design section.

The RefFrameItems identifies parts of the reference frame state using the following enumerations:

- Pos, the position,
- Vel, the velocity,
- Att, the attitude,
- Rate, the angular velocity, or attitude rate.

4.2.2 Building a Rooted Tree of Reference Frames

The most important utility of the Reference Frame Model lies in the association of reference frames to one another through the concept of a rooted tree. In short, a rooted tree of reference frames gives the following:

- Each reference frame is the child of a single parent reference frame, unless it is the root of the tree then its parent is NULL,
- Each reference frame, unless it is the root node, can have zero to many sibling reference frames that share the same parent, and
- Each reference frame can have zero to many child reference frames, to which the subject reference frame is the parent.

Each reference frame contains a member variable "state." This state is with respect to the reference frame's parent node, as described in the Conceptual Design and the Analysis sections.

The first action that should be taken is creating a reference frame and declaring it to be a root node of a tree. Creating the reference frame is trivial, and once it has been instantiated it can be made into a root node using the following member method:

```
virtual void make_root();
```

If this function is not called then it will not be possible to add children to this reference frame.

Next, the tree can be populated by adding children to either the root node or to already added reference frames. This is done using the RefFrame member function:

```
virtual void add_child(RefFrame & frame);
```

where the argument "frame" is the child to be added.

These actions result in a user defined rooted tree of reference frames, where the state of child reference frames can be defined. When defining the state, it is important to note that the attitude and attitude rates are over constrained in the state, meaning that there are redundent terms associated with those pieces of the state. For example, as noted in the analysis section, both the contained quaternion and the contained transformation matrix represent the attitude of the state. The RefFrameState class includes member functions that allows for the setting of one member, and then calculating the other member associated with this state property. These member functions must explicitly be called after setting a data member, and are as follows:

```
void compute_transformation ();

void compute_quaternion ();

void compute_ang_vel_unit ();

void compute_ang_vel_products ();
```

"compute_transformation" will compute the data member "T_parent_this" from a previously set "Q_parent_this." Conversely, "compute_quaternion" will do the opposite. "compute_ang_vel_unit" will calculate the angular velocity unit vector from a previously set "ang_vel_this," while "compute_ang_vel_products" will compute both the unit vector as well as the magnitude of the angular velocity.

Note that it is imperative to make sure all of these parameters associated with state are set, or the functions associated with calculating relative position and state of reference frames will not function properly.

Finally, there are functions used to manipulate the tree of reference frames. To remove a node from the tree, along with all of its children, call the following member function on the node to be removed:

```
virtual void remove_from_parent();
```

This will cause the removed node to be made into a root of a separate tree, with all of its children still present.

There are two functions used to move a node from one parent to another. They are:

```
virtual void transplant_node(RefFrame& new_parent);
virtual void reset_parent(RefFrame& new_parent);
```

Both of these functions re-organize the tree so that the argument "new_parent" is the parent of the invoking node. If "transplant_node" is invoked, then the state of the invoking node will be altered so that it still, after movement, has the same relative state to the rest of the tree as it did before invocation. If "reset_parent" is called, then the member variable "state" of the invoking reference frame will be unchanged, and the state of the reference frame with respect to its old parent will numerically match the state of the reference frame with respect to its new parent.

4.2.3 Calculating Relative States

Two functions exist for calculating the relative state of a reference frame with respect to an arbitrary reference frame, contained in the same tree as the subject reference frame. They are as follows:

```
virtual void compute_position_from(
   const RefFrame& in_frame, double rel_pos[3]) const;

virtual void compute_relative_state(
   const RefFrame& wrt_frame, const RefFrame & coordinate_frame,
   RefFrameState& rel_state) const;
```

"compute_position_from" calculates the position of the invoking reference frame, with respect to the supplied frame "in_frame," and in the coordinates of "in_frame." The resulting position is stored in the argument "rel_pos,"

"compute_relative_state" calculates the complete state, with respect to the supplied frame "wrt_frame," in the coordinates of "coordinate_frame," and stores it in the supplied state "rel_state," Per the usual pattern, the state will be as follows:

- The position of the invoking frame with respect to the argument "wrt_frame," in the argument "coordinate_frame" coordinates,
- The velocity for the invoking frame with respect to the argument "wrt_frame," in the argument "coordinate_frame" coordinates,
- The transformation from the argument "wrt_frame" to the invoking frame, and
- The angular velocity of the invoking frame, with respect to the argument "wrt_frame," in the invoking frame's coordinates.

Note that, while there are other functions available to the reference frame user to calculate specialized types of relative state, these are not the suggested method for doing so, and that any users of the Reference Frame Model should use the two member functions described above.

4.2.4 Using the Reference Frame Manager

JEOD supplies a manager tool for working with reference frames, which aids in many common interactions with the RefFrame class. This manager provides the following three key areas of functionality:

- Registering and Searching for RefFrames,
- Building a tree of RefFrames, and
- Accessing the RefFrame subscription functionality.

The following sections will explain these functionalities.

Registering and Searching for RefFrames

The RefFrameManager class gives utilities for regsitering RefFrames. This is done through the following function:

```
void add_ref_frame (RefFrame & ref_frame);
```

There are three requirements for the argument "ref_frame":

- The RefFrame must have a valid name, that is not a NULL pointer nor a single " 0" character,
- The RefFrame can not already be registered with the RefFrameManager, and
- The RefFrame must have a unique name, in that it can't duplicate an already registered name.

Failure to meet any of these requirements will result in an appropriately worded error message being sent to the JEOD Message Handler [8].

The RefFramemanager also contains functions to search for previously registered RefFrames, by name. These functions are:

```
RefFrame * find_ref_frame (const char * name) const;
RefFrame * find_ref_frame (const char * prefix, const char * suffix) const;
```

Both of these functions find reference frames by name. The first uses the name given, the second uses the dot-conjoined name:

```
"${prefix}.${suffix}"
```

as the search string. If the search fails, a NULL pointer is returned.

Building a RefFrame Tree

Two RefFrameManager functions can be used to assist in the building and track of a RefFrameTree. They are:

```
void add_frame_to_tree (RefFrame & ref_frame, RefFrame * parent);
void reset_tree_root_node ();
```

"add_frame_to_tree", as implied, adds a frame to the reference frame tree being maintained by this RefFrameManager. The inputs to the function are the RefFrame to be added, and the intended parent of this RefFrame. The given reference frame is either made a child of the given parent, or if NULL is given for the "parent" argument it is assumed to be the intended as the root and is made the internal root node for the RefFrameManager. Note that in the former case, the function "RefFrame::add_child()" will be called on the "parent" pointer, giving the "ref_frame" object as an argument. If it is the latter, then "RefFrame::make_root()" is called on "ref_frame".

The "reset_tree_root_node" function merely resets the internally tracked root of the RefFrame tree to NULL, preparing the RefFrameManager to have its tree re-built from the ground up.

Accessing RefFrame Subscription Functionality

Six functions exist for accessing the RefFrame subscription functionality through the RefFrameManager. They are:

```
void subscribe_to_frame (const char * frame_name);
void subscribe_to_frame (RefFrame & frame);

void unsubscribe_to_frame (const char * frame_name);
void unsubscribe_to_frame (RefFrame & frame);

bool frame_is_subscribed (const char * frame_name);
bool frame_is_subscribed (RefFrame & frame);
```

Each function does what its name purports; the two versions of "subscribe_to_frame" add a subscription, one taking the name of the frame and one taking a reference to the frame. If the name of the frame is supplied and this name does not exist in the registry of the RefFrameManager, then an error will be sent through the JEOD Message Handler [8].

Similarly, "unsubscribe_to_frame" takes either a name or a frame reference, with an error being sent if the named versions does not find a frame of the name in question.

Lastly, "frame_is_subscribed" will return true if a frame has any subscibers, false if not. Again, if the version taking a name as a paramter is used and the name given is not found in the registry, then an error will be sent.

4.3 Extension

There are three methods of extension for the Reference Frame Model: creating a reference frame owner class, extending the TreeLinksIterator classes, and extending the RefFrameManager class.

4.3.1 Creating a Reference Frame Owner

The Reference Frame Model includes an interface class for creating a reference frame owner. "Owning" a reference frame implies that either the RefFrame object is contained within the owning class, or that the owning class is solely responsible for updating the state of the RefFrame object.

In order to gain this interface, a class must inherit from the "RefFrameOwner" class contained within the Reference Frame Model. There are then three functions that may be overridden:

```
virtual void activate (void);

virtual void deactivate (void);

virtual bool is_active (void) const;
```

These functions will be utilized in the Dynamics Manager Model [5]. They should be overridden to add any activation / deactivation functionality associated with the particular class that is being written.

4.3.2 Extending the TreeLinksIterator Template Usage

While this section does not apply directly to the Reference Frame Model it is a useful utility class for traversing a rooted tree, and will be presented here.

The TreeLinksIterator class is a templatized class, intended to give a common interface for iterating over components of a rooted tree. The template takes two arguments, a class that contains the link information of the current node in the tree ("class Links"), and the class that contains ths class Links ("class Container"). For reference, an example of "class Links" is RefFrameLinks and an example of "class Container" is RefFrame, as it contains a member of type RefFrameLinks. From this point forward in this discussion, these will be referred to as "Links" and "Container", respectively.

In order to use the TreeLinksIterator templates, the Links class must be set up as a non-cyclical, rooted tree, where each node in the tree has a parent (possibly none if it is a root), possible sibling nodes and possible child nodes. These linkages must be represented through the implementation in the class of the following data members:

```
// A reference to the containing class
Container& container_;
```

```
// A pointer to the Links object of the parent. NULL if this is the root.
Links* parent_;

// The array of Links object of the node's direct children. NULL if there are
// no children.
Links* child_head_;

// A pointer to the next Links object in the forward list of all the child
// links of the current Links object's parent. NULL if this particular node
// has no siblings, or if it is the last sibling in the list
Links* next_;

// An array of Links objects pointers, starting at the root node and following
// the most direct path to the Links of this object. Ends with a pointer to
// 'this'
Links** path_to_node;

// The length of path_to_node
unsigned int path_length;
```

These values must be properly implemented in the Links class for the TreeLinksIterator to work as intended. They must also be updated if there are any changes to the structure of the tree (addition, subtraction, or moving of nodes).

For encapsulation purposes, it is possible for these attributes to be made protected or private members of the Links class. If this is desired, then the TreeLinksIterator templates can be made a friend of the Links class. An example of this can be seen in the implementation for RefFrameLinks, where the following friend classes are declared:

```
friend class TreeLinksIterator<RefFrameLinks, RefFrame>;
friend class TreeLinksParentIterator<RefFrameLinks, RefFrame>;
friend class TreeLinksDescentIterator<RefFrameLinks, RefFrame>;
friend class TreeLinksChildIterator<RefFrameLinks, RefFrame>;

friend class TreeLinksIterator<const RefFrameLinks, const RefFrame>;
friend class TreeLinksParentIterator<const RefFrameLinks, const RefFrame>;
friend class TreeLinksDescentIterator<const RefFrameLinks, const RefFrame>;
friend class TreeLinksChildIterator<const RefFrameLinks, const RefFrame>;
```

The TreeLinksIterator may now be used with the created Links and Container classes. There are three specific iterators available, with differing functionality:

- TreeLinksParentIterator, an iterator that steps up the tree to parents of nodes until it finds the root,
- TreeLinksDescentIterator, an iterator that steps down the tree through the most direct path to a specified node, and

- TreeLinksChildIterator, an iterator that steps through all of the children of a specified node.

A TreeLinksParent iterator can be created using one of two constructors:

```
TreeLinksParentIterator<Links,Container>::TreeLinksParentIterator (
   Links& start);

TreeLinksParentIterator<Links,Container>::TreeLinksParentIterator (
   Links* start);
```

Where the start parameter in both cases indicates the initial Links object the iterator will be pointing to.

A TreeLinksDescentIterator can be created using one of two constructors:

```
TreeLinksDescentIterator<Links,Container>::TreeLinksDescentIterator (
   Links* ref_links,
   unsigned int start_index);

TreeLinksDescentIterator<Links,Container>::TreeLinksDescentIterator (
   Links& ref_links,
   unsigned int start_index);
```

The "ref_links" argument corresponds to the Links object, in the tree, that the descent iterator is driving towards. It will stop at this Links object and not continue. The "start_index" parameter corresponds to which Links object, contained in the "path_to_node" array of "ref_links," the iterator should initially point to.

A TreeLinksChildIterator can be created using one of two constructors:

```
TreeLinksChildIterator<Links,Container>::TreeLinksChildIterator (
   Links* parent_links);

TreeLinksChildIterator<Links,Container>::TreeLinksChildIterator (
   Links& parent_links);
```

The argument "parent_links" is the parent of the children which the iterator will step through.

The iterator may then be used in a for loop using the following example:

```
for (TreeLinksIteratorType <LinksClass, ContainerClass> iter (init);
     ! iter.done ();  // or !iter.at(target);
     iter.next()){
   ContainerClass* node = iter.current();
   loop body
}
```

Note that the ending condition for the for loop can have two forms. The "done" member functions returns false once the iteration has run to completion (Parent iterator goes past the root node, Descent iterator descends past the indicated node, Children iterator goes through all children). The "at" function, on the other hand, will stop the iteration once the iterator has reached the node specified by the "target" argument.

Care must be taken when the "at" function is used. If the ending node specified by the "target" argument is never found, it leaves the possibility of NULL being returned by the "iter.current()" function, a condition that must be checked for by the user.

4.3.3 Extending the RefFrameManager Class

The RefFrameManager class can be inherited from for any class which requires its functionality. When doing this, a user should be aware of a few features and warnings associated with internal usage of the RefFrameManager.

There is a protected utility method that users can optionally use in their own class. Its signature is as follows:

```
bool validate_name (
   const char * file,
   unsigned int line,
   const char * variable_value,
   const char * variable_type,
   const char * variable_name) const;
```

This function will check if a given name (the "variable_value" argument) is either NULL, or if it contains only the single character "\0". If either of these is the case, then the function will return "false" and an error message will be sent through the JEOD Message Handler [8]. This is a useful utility function, and its full API can be found in the *Reference Manual* [1].

Second, there is one virtual function that can be overridden by the extender. It is the "add_ref_frame" function. It is possible that an extender of this class would like to extend this function in an inheriting class. This is possible by overriding the function, then calling the RefFrameManager's version of the function directly from the inheriting method. An explicit example of this can be found in the EphemerisManager class [9].

Chapter 5

Verification and Validation

5.1 Verification

Inspection refframes_1: Top-level inspection

This document structure, the code, and associated files have been inspected, and together satisfy Requirement refframes_1.

Inspection refframes_2: Reference Frame Representation

By inspection, the Reference Frame Model satisfies the requirements set forth in Requirement refframes_2

Inspection refframes_3: Reference Frame State

The following code exists in the file "ref_frame_state.hh":

```
class RefFrameTrans {


 public:

   double position[3]; /* M
      Position of the subject reference frame origin with respect to the parent
      frame origin and expressed in parent reference frame coordinates */

   double velocity[3]; /* M/s
      Velocity of the subject reference frame origin with respect to the parent
      frame origin and expressed in parent reference frame coordinates */
```

```
};

class RefFrameRot {

 public:
   Quaternion  Q_parent_this; /* --
      Left transformation quaternion from the parent reference frame to the
      subject reference frame */

   double T_parent_this[3][3]; /* --
      Transformation matrix from the parent reference frame to the
      subject reference frame */

   double ang_vel_this[3]; /* r/s
      Angular velocity of the subject reference frame with respect to the
      parent reference frame expressed in subject reference frame coordinates */

   double ang_vel_mag; /* r/s
      Magnitude of ang_vel_this */

   double ang_vel_unit[3]; /* --
      Unit vector in the direction of ang_vel_this */

};
```

These parameters represent the translational and rotational states of an object in Cartesian three space, which satisfies Requirement refframes_3.

Inspection refframes_4: Reference Frame Tree Structure

The file "ref_frame_links.hh" contains the following entities:

```
class RefFrameLinks {

 // Member data
 private:

   RefFrame & container_; /* --
      The object to which this set of links pertains; the container. */

   RefFrameLinks * parent_; /* --
     The RefFrameLinks object that is the immediate parent of this RefFrameLinks
     object in the directed tree that contains this RefFrameLinks object.
     This pointer is NULL for all root objects. */
```

```
RefFrameLinks * child_head_; /* --
   The RefFrameLinks object that represents the first RefFrameLinks object
   that is a direct child of this RefFrameLinks object.
   The child_head_ pointer in the parent and the next_ pointers in the
   child objects collectively enumerate all of the child objects.
   This pointer is NULL for all atomic (childless) objects. */

RefFrameLinks * child_tail_; /* --
   The RefFrameLinks object that represents the last RefFrameLinks object
   that is a direct child of this RefFrameLinks object.
   The child_tail_ pointer in the parent and the prev_ pointers in the
   child objects collectively enumerate all of the child objects, but in a
   reverse sense from the forward head/next links.
   This pointer is NULL for all atomic (childless) objects. */

RefFrameLinks * next_; /* --
   The RefFrameLinks object that represents the next RefFrameLinks object
   in the forward list of all the child links of some parent node.
   This pointer is NULL for RefFrameLinks nodes that have no siblings and for
   the last child node (the child_tail_) in the forward sequence. */

RefFrameLinks * prev_; /* --
   The RefFrameLinks object that represents the previous RefFrameLinks object
   in the reverse list of all the child links of some parent node.
   This pointer is NULL for RefFrameLinks nodes that have no siblings and for
   the last child node (the child_head_) in the reverse sequence. */

RefFrameLinks ** path_to_node_; /* --
   Array of pointers to RefFrameLinks nodes containing the sequence of links
   from the root node of the tree to this RefFrameLinks object.
   The path_to_node_ does not exist until the links object is made viable
   by either a call to attach() or to make_root().
   The zeroth element of this array is the root object.
   The last element is this node. */

unsigned int path_size_; /* --
   The allocated size of the path_to_node_ array. */

unsigned int path_length_; /* --
   The active length of the path_to_node_ array. */
```

These components represent the rooted tree structure associated with the Reference Frame Model, which fullfills Requirement refframes_4.

Inspection refframes_5: Reference Frame Active/Inactive Functionality

The file "ref_frame.hh" contains the following items:

```
class RefFrame {

 protected:


   bool active; /* --
      Flag indicating whether the frame is actively being updated. */
 // Member functions

  public:

   // activate: Mark the frame as active
   virtual void activate (void);

   // deactivate: Mark the frame as inactive
   virtual void deactivate (void);

   // is_active: Is the frame active?
   virtual bool is_active (void) const;

};
```

These parameters fullfill the requirements set forth in Requirement refframes_6.

Inspection refframes_6: Reference Frame Subscription Functionality

The file "ref_frame.hh" contains the following items:

```
class RefFrame {

   unsigned int subscribers; /* --
     Number of subscribers for this reference frame.
     Note: The subscription mechanism is intended primarily for use with
     ephemeris-based reference frames. */

 public:

   // subscribe: Increment the subscription count and activate
   virtual void subscribe (void);

   // unsubscribe: Decrement the subscription count and deactivate when unsed
```

```
   virtual void unsubscribe (void);

   // subscriptions: Return the subscription count
   virtual unsigned int subscriptions (void) const;

};
```

These parameters fullfil the requirements of Requirement refframes_7.

Inspection refframes_7: Reference Frame Manager Utility

The class "RefFrameManager" contains the following implemented functions:

```
// Find a reference frame.
RefFrame * find_ref_frame (const char * name) const;
RefFrame * find_ref_frame (const char * prefix, const char * suffix) const;
```

These functions allow a user to search the reference frames registered with this particular RefFrameManager. They satisfy Requirement 8.1, and partially satisfy Requirement refframes_8.

The class "RefFrameManager" contains the following implemented functions:

```
// Add a reference frame to the list of such.
virtual void add_ref_frame (RefFrame & ref_frame);

// Add a reference frame to the reference frame tree.
void add_frame_to_tree (RefFrame & ref_frame, RefFrame * parent);

// Reset the root node in anticipation of rebuilding the entire tree.
void reset_tree_root_node ();
```

These functions allow for the building, or rebuilding, of a reference frame tree. They satisfy Requirement 8.2, and partially satisfy Requirement refframes_8.

The class "RefFrameManager" contains the following implemented functions:

```
// Add a subscription to a reference frame.
void subscribe_to_frame (const char * frame_name);
void subscribe_to_frame (RefFrame & frame);

// Remove a subscription from a reference frame.
void unsubscribe_to_frame (const char * frame_name);
```

```
void unsubscribe_to_frame (RefFrame & frame);

// Check whether a reference frame has subscriptions.
bool frame_is_subscribed (const char * frame_name);
bool frame_is_subscribed (RefFrame & frame);
```

These functions allow subscribing to a reference frame, unsubscribing to a reference frame, and checking the subscription status of a reference frame. They satisfy Requirement 8.3, and partially satisfy Requirement refframes_8.

5.2 Validation

The goal of this section is to validate the relative state computations, of both the full state object as well as the position only version, of the Reference Frame Model. To do this, two configurations of reference frames will be used.

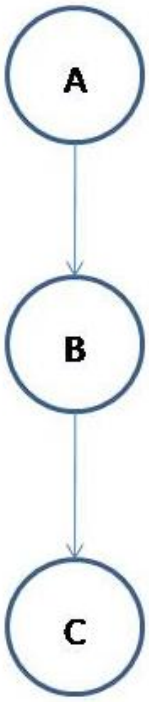

Figure 5.1: Reference Frame A is a parent to Reference Frame B, who is a parent to Reference Frame C

The first is shown above in Figure 5.1. As shown, this configuration contains three reference frames: frame A, which is a parent to frame B, which is a parent to frame C.

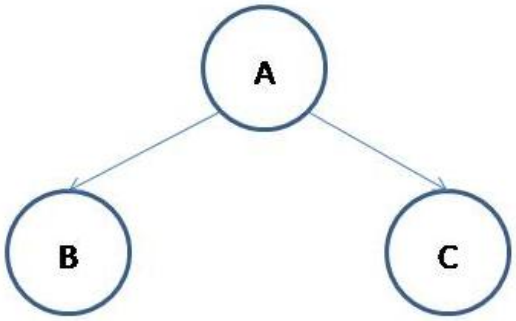

Figure 5.2: Reference Frame A is a parent to both Reference Frame B and Reference Frame C

The second configuration is shown above in Figure 5.2. As shown, this configuration contains three reference frames: Frame A, which is a parent to both Frame B and Frame C. Frame B and C are, thus, siblings of one another.

The six tests that follow will use these two configurations of reference frame trees. Three tests will involve the first configuration. A random relative state of Frame B with respect to A will be specified, and a random relative state of Frame C with respect to B will be specified. The reference frame model will then calculate the relative states of Frame A with respect to C, and of Frame C with respect to Frame A. This will be done both for the complete state, as well as the position specific version of the algorithm. These results will then be compared against an independently programmed Matlab solution.

Similarly, three tests will involve the second configuration. A random relative state of Frame B with respect to A will be specified, and a random relative state of Frame C with respect to A will be specified. The reference frame model will then calculate the relative state of Frame B with respect to C, and of Frame C with respect to Frame A. This will be done both for the complete state, as well as the position specific version of the algorithm. These results will then be compared against an independently programmed Matlab solution.

Note that all random relative states were created in Matlab, and the states themselves will be presented with each test.

Test refframes_1: Relative State Test 1, Configuration 1

Purpose:

SIM directory: SIM_REF_FRAMES RUN directory: SET_test_val/RUN1_ref_frame_ver

The purpose of this test is to demonstrate the relative state and position calculations associated with Reference Frame Model.

Requirements:

By passing this test, the Reference Frame Model partially satisfies the Requirement ref-frames_5

Procedure:

The following relative states were inputted for reference Frames B and C, configured as shown in 5.1.

$S_{A:B}$**:**

$$x_{A:B}: \begin{pmatrix} 1848.16 \\ 9048.81 \\ 9797.48; \end{pmatrix}$$

$$v_{A:B}: \begin{pmatrix} -408.72 \\ -594.896 \\ 262.212; \end{pmatrix}$$

$$T_{A:B}: \begin{pmatrix} 0.02165177281818631 & -0.9708670732541415 & 0.2386384855901647 \\ -0.1303430129048092 & 0.2339159463811323 & 0.9634801653461872 \\ -0.9912325154678525 & -0.05196591286176722 & -0.1214810445284236 \end{pmatrix}$$

$$w_{A:B}: \begin{pmatrix} 11.7418 \\ -29.6676 \\ 31.8778 \end{pmatrix}$$

$S_{B:C}$**:**

$$x_{B:C}: \begin{pmatrix} -2372.84 \\ -4588.49 \\ 9630.89 \end{pmatrix}$$

$$v_{B:C}: \begin{pmatrix} 488.898 \\ 624.06 \\ -679.136 \end{pmatrix}$$

$$T_{B:C}: \begin{pmatrix} 0.7059970877834382 & -0.6729909535450022 & -0.2205703708295675 \\ 0.3550094808358443 & 0.06680598428211491 & 0.9324726424838222 \\ -0.6128102320930751 & -0.73662754286728 & 0.2860829993749103 \end{pmatrix}$$

$$w_{B:C}: \begin{pmatrix} -3.77389 \\ 88.5168 \\ 91.3287 \end{pmatrix}$$

The following relative states will then be calculated:

$S_{A:C}$ $S_{C:A}$

Additionally, the relative positions:

$x_{A:C}$ $x_{C:A}$

will be calculated using the separate, position specific algorithm.

These relative states and positions will then be compared to an independently programmed Matlab relative state and position calculation.

Results:

The following are the results from the independently programmed Matlab relative state solution:

$$x_{A:C}: \begin{pmatrix} -7151.589902224503 \\ 9778.723254768729 \\ 3640.339375024558 \end{pmatrix}$$

$$v_{A:C}: \begin{pmatrix} 144957.2752796087 \\ 96815.65534740115 \\ -198952.3536598123 \end{pmatrix}$$

$$T_{A:C}: \begin{pmatrix} 0.3216422806127932 & -0.8313905014762567 & -0.4531402402979231 \\ -0.9253183116582367 & -0.3774968226845905 & 0.03580741505594964 \\ -0.2008289457029322 & 0.4077817834554374 & -0.8907197941270596 \end{pmatrix}$$

$$w_{A:C}: \begin{pmatrix} 17.45051485149649 \\ 120.428453505361 \\ 115.1068727450524 \end{pmatrix}$$

$$x_{C:A}: \begin{pmatrix} 12079.77557590324 \\ -3056.411278357381 \\ -2181.30913023573 \end{pmatrix}$$

$$v_{C:A}: \begin{pmatrix} -145408.3486105788 \\ -1250726.964369239 \\ 1320505.798372557 \end{pmatrix}$$

$$T_{C:A}: \begin{pmatrix} 0.3216422806127932 & -0.9253183116582367 & -0.2008289457029322 \\ -0.8313905014762567 & -0.3774968226845905 & 0.4077817834554374 \\ -0.4531402402979231 & 0.03580741505594964 & -0.8907197941270596 \end{pmatrix}$$

$$w_{C:A}: \begin{pmatrix} 128.9386217750408 \\ 13.03106499654173 \\ 106.1232688679998 \end{pmatrix}$$

The following are the results from the Reference Frame Model full relative state calculation:

$$x_{A:C}: \begin{pmatrix} -7151.589902224503 \\ 9778.723254768729 \\ 3640.339375024558 \end{pmatrix}$$

$$v_{A:C}: \begin{pmatrix} 144957.2752796087 \\ 96815.65534740115 \\ -198952.3536598123 \end{pmatrix}$$

$$T_{A:C}: \begin{pmatrix} 0.3216422806127932 & -0.8313905014762567 & -0.4531402402979231 \\ -0.9253183116582367 & -0.3774968226845905 & 0.03580741505594964 \\ -0.2008289457029322 & 0.4077817834554374 & -0.8907197941270596 \end{pmatrix}$$

$$w_{A:C}: \begin{pmatrix} 17.45051485149649 \\ 120.428453505361 \\ 115.1068727450524 \end{pmatrix}$$

$$x_{C:A}: \begin{pmatrix} 12079.77557590324 \\ -3056.411278357381 \\ -2181.30913023573 \end{pmatrix}$$

$$v_{C:A}: \begin{pmatrix} -145408.3486105788 \\ -1250726.964369239 \\ 1320505.798372557 \end{pmatrix}$$

$$T_{C:A}: \begin{pmatrix} 0.3216422806127932 & -0.9253183116582367 & -0.2008289457029322 \\ -0.8313905014762567 & -0.3774968226845905 & 0.4077817834554374 \\ -0.4531402402979231 & 0.03580741505594964 & -0.8907197941270596 \end{pmatrix}$$

$$w_{C:A}: \begin{pmatrix} 128.9386217750408 \\ 13.03106499654173 \\ 106.1232688679998 \end{pmatrix}$$

The following are the results from the Reference Frame Model position specific calculation:

$$x_{A:C}: \begin{pmatrix} -7151.589902224503 \\ 9778.723254768729 \\ 3640.339375024558 \end{pmatrix}$$

$x_{C:A}$: $\begin{pmatrix} 12079.77557590324 \\ -3056.411278357381 \\ -2181.30913023573 \end{pmatrix}$

All Reference Frame Model produced states are a numerical match to the independently programmed Matlab solution.

Test refframes_2: Relative State Test 2, Configuration 1

Purpose:

SIM directory: SIM_REF_FRAMES RUN directory: SET_test_val/RUN2_ref_frame_ver

The purpose of this test is to demonstrate the relative state and position calculations associated with Reference Frame Model.

Requirements:

By passing this test, the Reference Frame Model partially satisfies the Requirement ref-frames_5

Procedure:

$x_{A:B}$: $\begin{pmatrix} -8033.64 \\ 604.712 \\ -3992.58 \end{pmatrix}$

$v_{A:B}$: $\begin{pmatrix} 627.973 \\ -291.984 \\ 431.651 \end{pmatrix}$

$T_{A:B}$: $\begin{pmatrix} 0.4272687159464157 & -0.3061054736119365 & -0.8507296182679391 \\ 0.1149180951186981 & 0.9516978685416249 & -0.2847191571138446 \\ 0.8967916568455525 & 0.02388736147296944 & 0.4418077841936113 \end{pmatrix}$

$w_{A:B}$: $\begin{pmatrix} 10.6216 \\ 37.241 \\ -19.8118 \end{pmatrix}$

$S_{B:C}$**:**

$x_{B:C}$: $\begin{pmatrix} -4177.44 \\ -9830.52 \\ -3014.55 \end{pmatrix}$

$$v_{B:C}: \begin{pmatrix} 698.106 \\ -666.528 \\ 178.132 \end{pmatrix}$$

$$T_{B:C}: \begin{pmatrix} 0.6361951982813826 & 0.3331380394803984 & 0.695898495712463 \\ 0.564199301993485 & 0.414337780792221 & -0.7141451890464865 \\ -0.5262459665534518 & 0.8469611856864128 & 0.07564345726425621 \end{pmatrix}$$

$$w_{B:C}: \begin{pmatrix} -3.26008 \\ 56.12 \\ 88.1867 \end{pmatrix}$$

The following relative states will then be calculated:

$S_{A:C}$ $S_{C:A}$

Additionally, the relative positions:

$x_{A:C}$ $x_{C:A}$

will be calculated using the separate, position specific algorithm.

These relative states and positions will then be compared to an independently programmed Matlab relative state and position calculation.

Results:

The following are the results from the independently programmed Matlab relative state solution:

$$x_{A:C}: \begin{pmatrix} -13651.65734631322 \\ -7544.245326498707 \\ 1028.37764908716 \end{pmatrix}$$

$$v_{A:C}: \begin{pmatrix} -71105.87066818211 \\ 203306.0472083863 \\ 251222.1995609853 \end{pmatrix}$$

$$T_{A:C}: \begin{pmatrix} 0.9341858593359342 & 0.1389271085396476 & -0.3286275075668436 \\ -0.3517598275083078 & 0.2045608440130484 & -0.9134658640846932 \\ -0.05968085093317921 & 0.9689448485901985 & 0.239966823586948 \end{pmatrix}$$

$$w_{A:C}: \begin{pmatrix} 2.116722828998875 \\ 91.69155425688828 \\ 112.6401743111756 \end{pmatrix}$$

$$x_{C:A}: \begin{pmatrix} 14139.23862240651 \\ -2319.459564082803 \\ 6248.438600636182 \end{pmatrix}$$

$$v_{C:A}: \begin{pmatrix} -713453.4736831919 \\ -1416537.83223535 \\ 1039837.419066911 \end{pmatrix}$$

$$T_{C:A}: \begin{pmatrix} 0.9341858593359342 & -0.3517598275083078 & -0.05968085093317921 \\ 0.1389271085396476 & 0.2045608440130484 & 0.9689448485901985 \\ -0.3286275075668436 & -0.9134658640846932 & 0.239966823586948 \end{pmatrix}$$

$$w_{C:A}: \begin{pmatrix} 36.99845422653996 \\ -128.1926885529855 \\ 57.42281334830777 \end{pmatrix}$$

The following are the results from the Reference Frame Model full relative state calculation:

$$x_{A:C}: \begin{pmatrix} -13651.65734631322 \\ -7544.245326498707 \\ 1028.37764908716 \end{pmatrix}$$

$$v_{A:C}: \begin{pmatrix} -71105.87066818211 \\ 203306.0472083863 \\ 251222.1995609853 \end{pmatrix}$$

$$T_{A:C}: \begin{pmatrix} 0.9341858593359342 & 0.1389271085396476 & -0.3286275075668436 \\ -0.3517598275083078 & 0.2045608440130484 & -0.9134658640846932 \\ -0.05968085093317921 & 0.9689448485901985 & 0.239966823586948 \end{pmatrix}$$

$$w_{A:C}: \begin{pmatrix} 2.116722828998875 \\ 91.69155425688828 \\ 112.6401743111756 \end{pmatrix}$$

$$x_{C:A}: \begin{pmatrix} 14139.23862240651 \\ -2319.459564082803 \\ 6248.438600636182 \end{pmatrix}$$

$$v_{C:A}: \begin{pmatrix} -713453.4736831919 \\ -1416537.83223535 \\ 1039837.419066911 \end{pmatrix}$$

$$T_{C:A}: \begin{pmatrix} 0.9341858593359342 & -0.3517598275083078 & -0.05968085093317921 \\ 0.1389271085396476 & 0.2045608440130484 & 0.9689448485901985 \\ -0.3286275075668436 & -0.9134658640846932 & 0.239966823586948 \end{pmatrix}$$

$$w_{C:A}: \begin{pmatrix} 36.99845422653996 \\ -128.1926885529855 \\ 57.42281334830777 \end{pmatrix}$$

The following are the results from the Reference Frame Model position specific calculation:

$$x_{A:C}: \begin{pmatrix} -13651.65734631322 \\ -7544.245326498707 \\ 1028.37764908716 \end{pmatrix}$$

$$x_{C:A}: \begin{pmatrix} 14139.23862240651 \\ -2319.459564082803 \\ 6248.438600636182 \end{pmatrix}$$

All Reference Frame Model produced states are a numerical match to the independently programmed Matlab solution.

Test refframes_3: Relative State Test 3, Configuration 1

Purpose:

SIM directory: SIM_REF_FRAMES RUN directory: SET_test_val/RUN3_ref_frame_ver

The purpose of this test is to demonstrate the relative state and position calculations associated with Reference Frame Model.

Requirements:

By passing this test, the Reference Frame Model partially satisfies the Requirement ref-frames_5

Procedure:

$$x_{A:B}: \begin{pmatrix} 2664.71 \\ -1536.57 \\ 2810.05 \end{pmatrix}$$

$$v_{A:B}: \begin{pmatrix} -875.372 \\ -518.052 \\ 943.623 \end{pmatrix}$$

$$T_{A:B}: \begin{pmatrix} 0.8532057436490301 & 0.2501060894816011 & -0.4576973923986534 \\ -0.3517114947196397 & 0.9238850041384419 & -0.1507830348885044 \\ 0.3851480020542523 & 0.2896263854215159 & 0.8762291785721845 \end{pmatrix}$$

$$w_{A:B}: \begin{pmatrix} -67.6122 \\ 28.9065 \\ 67.1808 \end{pmatrix}$$

$S_{B:C}$**:**

$$x_{B:C}: \begin{pmatrix} 6021.7 \\ 3867.71 \\ 9159.91 \end{pmatrix}$$

$$v_{B:C}: \begin{pmatrix} -460.916 \\ 770.16 \\ 322.472 \end{pmatrix}$$

$$T_{B:C}: \begin{pmatrix} -0.9655570135702486 & -0.1962017385355807 & -0.1708933332254907 \\ 0.1197804372000868 & 0.2478810463059914 & -0.9613571832292113 \\ 0.2309811689498931 & -0.9487148489814329 & -0.2158421527706723 \end{pmatrix}$$

$$w_{B:C}: \begin{pmatrix} 17.5874 \\ 72.1758 \\ -47.3486 \end{pmatrix}$$

The following relative states will then be calculated:

$S_{A:C}$ $S_{C:A}$

Additionally, the relative positions:

$x_{A:C}$ $x_{C:A}$

will be calculated using the separate, position specific algorithm.

These relative states and positions will then be compared to an independently programmed Matlab relative state and position calculation.

Results:

The following are the results from the independently programmed Matlab relative state solution:

$$x_{A:C}: \begin{pmatrix} 9970.061996786033 \\ 6195.764732474048 \\ 7496.92897541955 \end{pmatrix}$$

$$v_{A:C}: \begin{pmatrix} -525060.1334150368 \\ 821188.5121990081 \\ -536983.4183932868 \end{pmatrix}$$

$$T_{A:C}: \begin{pmatrix} -0.8206316089279889 & -0.472254751249236 & 0.3217750959144952 \\ -0.3552500546942768 & -0.01946300784092403 & -0.9345686651956066 \\ 0.4476172037220795 & -0.8812472077960342 & -0.1517965667718475 \end{pmatrix}$$

$$w_{A:C}: \begin{pmatrix} 65.71857751718055 \\ 6.657809933899429 \\ -104.8902192702117 \end{pmatrix}$$

$$x_{C:A}: \begin{pmatrix} 8695.402309937857 \\ 10668.84819285635 \\ 2135.237178517832 \end{pmatrix}$$

$$v_{C:A}: \begin{pmatrix} -1003556.704160732 \\ 379994.6857444195 \\ 233934.6017263197 \end{pmatrix}$$

$$T_{C:A}: \begin{pmatrix} -0.8206316089279889 & -0.3552500546942768 & 0.4476172037220795 \\ -0.472254751249236 & -0.01946300784092403 & -0.8812472077960342 \\ 0.3217750959144952 & -0.9345686651956066 & -0.1517965667718475 \end{pmatrix}$$

$$w_{C:A}: \begin{pmatrix} 103.2465959950725 \\ -61.26872137221126 \\ -30.84639621406894 \end{pmatrix}$$

The following are the results from the Reference Frame Model full relative state calculation:

$$x_{A:C}: \begin{pmatrix} 9970.061996786033 \\ 6195.764732474048 \\ 7496.92897541955 \end{pmatrix}$$

$$v_{A:C}: \begin{pmatrix} -525060.1334150368 \\ 821188.5121990081 \\ -536983.4183932868 \end{pmatrix}$$

$$T_{A:C}: \begin{pmatrix} -0.8206316089279889 & -0.472254751249236 & 0.3217750959144952 \\ -0.3552500546942768 & -0.01946300784092403 & -0.9345686651956066 \\ 0.4476172037220795 & -0.8812472077960342 & -0.1517965667718475 \end{pmatrix}$$

$$w_{A:C}: \begin{pmatrix} 65.71857751718055 \\ 6.657809933899429 \\ -104.8902192702117 \end{pmatrix}$$

$$x_{C:A}: \begin{pmatrix} 8695.402309937857 \\ 10668.84819285635 \\ 2135.237178517832 \end{pmatrix}$$

$$v_{C:A}: \begin{pmatrix} -1003556.704160732 \\ 379994.6857444195 \\ 233934.6017263197 \end{pmatrix}$$

$$T_{C:A}: \begin{pmatrix} -0.8206316089279889 & -0.3552500546942768 & 0.4476172037220795 \\ -0.472254751249236 & -0.01946300784092403 & -0.8812472077960342 \\ 0.3217750959144952 & -0.9345686651956066 & -0.1517965667718475 \end{pmatrix}$$

$$w_{C:A}: \begin{pmatrix} 103.2465959950725 \\ -61.26872137221126 \\ -30.84639621406894 \end{pmatrix}$$

The following are the results from the Reference Frame Model position specific calculation:

$$x_{A:C}: \begin{pmatrix} 9970.061996786033 \\ 6195.764732474048 \\ 7496.92897541955 \end{pmatrix}$$

$$x_{C:A}: \begin{pmatrix} 8695.402309937857 \\ 10668.84819285635 \\ 2135.237178517832 \end{pmatrix}$$

All Reference Frame Model produced states are a numerical match to the independently programmed Matlab solution.

Test reffframes_4: Relative State Test 1, Configuration 2

Purpose:

SIM directory: SIM_REF_FRAMES RUN directory: SET_test_val/RUN1_ref_frame_ver

The purpose of this test is to demonstrate the relative state and position calculations associated with Reference Frame Model.

Requirements:

By passing this test, the Reference Frame Model partially satisfies the Requirement ref-frames_5

Procedure:

$x_{A:B}$: $\begin{pmatrix} -7790.52 \\ 7150.37 \\ -9037.21 \end{pmatrix}$

$v_{A:B}$: $\begin{pmatrix} -197.81 \\ 30.5409 \\ -744.074 \end{pmatrix}$

$T_{A:B}$: $\begin{pmatrix} 0.9055697770991294 & 0.4006122484928437 & 0.1394747473994526 \\ -0.4100036383807383 & 0.9109467042835068 & 0.04553151073239373 \\ -0.1088135805825256 & -0.09841711392093039 & 0.9891782834091561 \end{pmatrix}$

$w_{A:B}$: $\begin{pmatrix} -60.9867 \\ -61.7666 \\ 85.9442 \end{pmatrix}$

$S_{A:C}$**:**

$x_{A:C}$: $\begin{pmatrix} -5004.72 \\ 4710.88 \\ 596.189 \end{pmatrix}$

$v_{A:C}$: $\begin{pmatrix} -521.65 \\ -96.73 \\ 818.149 \end{pmatrix}$

$T_{A:C}$: $\begin{pmatrix} 0.09641301427589605 & -0.8301616645135473 & 0.5491230658516651 \\ -0.6488031425776438 & 0.3659638227289848 & 0.6671768600865603 \\ -0.7548238290221164 & -0.4205973029225039 & -0.5033278215186462 \end{pmatrix}$

$w_{B:C}$: $\begin{pmatrix} -65.9605 \\ -51.8595 \\ 97.2975 \end{pmatrix}$

The following relative states will then be calculated:

$S_{B:C}$ $S_{C:B}$

Additionally, the relative positions:

$x_{A:B}$ $x_{B:A}$

will be calculated using the separate, position specific algorithm.

These relative states and positions will then be compared to an independently programmed Matlab relative state and position calculation.

Results:

The following are the results from the independently programmed Matlab relative state solution:

$$x_{B:C}: \begin{pmatrix} 2889.062603090087 \\ -2925.810301475702 \\ 9466.103778667652 \end{pmatrix}$$

$$v_{B:C}: \begin{pmatrix} 333106.2646036395 \\ -825516.6365463276 \\ -355290.0084029901 \end{pmatrix}$$

$$T_{B:C}: \begin{pmatrix} -0.168675418285182 & -0.7707603161852675 & 0.6143916814706279 \\ -0.3478726032834237 & 0.6297627576170896 & 0.6945383509953449 \\ -0.9222435985959093 & -0.09657848679962694 & -0.3743518942604159 \end{pmatrix}$$

$$w_{B:C}: \begin{pmatrix} -176.6581028286698 \\ -93.86834069564757 \\ 67.26095563344904 \end{pmatrix}$$

$$x_{C:B}: \begin{pmatrix} -7583.680047586309 \\ -3726.980016279078 \\ 5925.503040873298 \end{pmatrix}$$

$$v_{C:B}: \begin{pmatrix} -56264.49890069378 \\ 345818.1552020046 \\ 147940.7284172714 \end{pmatrix}$$

$$T_{C:B}: \begin{pmatrix} -0.168675418285182 & -0.3478726032834237 & -0.9222435985959093 \\ -0.7707603161852675 & 0.6297627576170896 & -0.09657848679962694 \\ 0.6143916814706279 & 0.6945383509953449 & -0.3743518942604159 \end{pmatrix}$$

$$w_{C:B}: \begin{pmatrix} -0.421117663390973 \\ -70.55030878770818 \\ 198.9116975508892 \end{pmatrix}$$

The following are the results from the Reference Frame Model full relative state calculation:

$$x_{B:C}: \begin{pmatrix} 2889.062603090087 \\ -2925.810301475702 \\ 9466.103778667652 \end{pmatrix}$$

$$v_{B:C}: \begin{pmatrix} 333106.2646036395 \\ -825516.6365463276 \\ -355290.0084029901 \end{pmatrix}$$

$$T_{B:C}: \begin{pmatrix} -0.168675418285182 & -0.7707603161852675 & 0.6143916814706279 \\ -0.3478726032834237 & 0.6297627576170896 & 0.6945383509953449 \\ -0.9222435985959093 & -0.09657848679962694 & -0.3743518942604159 \end{pmatrix}$$

$$w_{B:C}: \begin{pmatrix} -176.6581028286698 \\ -93.86834069564757 \\ 67.26095563344904 \end{pmatrix}$$

$$x_{C:B}: \begin{pmatrix} -7583.680047586309 \\ -3726.980016279078 \\ 5925.503040873298 \end{pmatrix}$$

$$v_{C:B}: \begin{pmatrix} -56264.49890069378 \\ 345818.1552020046 \\ 147940.7284172714 \end{pmatrix}$$

$$T_{C:B}: \begin{pmatrix} -0.168675418285182 & -0.3478726032834237 & -0.9222435985959093 \\ -0.7707603161852675 & 0.6297627576170896 & -0.09657848679962694 \\ 0.6143916814706279 & 0.6945383509953449 & -0.3743518942604159 \end{pmatrix}$$

$$w_{C:B}: \begin{pmatrix} -0.421117663390973 \\ -70.55030878770818 \\ 198.9116975508892 \end{pmatrix}$$

The following are the results from the Reference Frame Model position specific calculation:

$$x_{B:C}: \begin{pmatrix} 2889.062603090087 \\ -2925.810301475702 \\ 9466.103778667652 \end{pmatrix}$$

$$x_{C:B}: \begin{pmatrix} -7583.680047586309 \\ -3726.980016279078 \\ 5925.503040873298 \end{pmatrix}$$

All Reference Frame Model produced states are a numerical match to the independently programmed Matlab solution.

Test reffframes_5: Relative State Test 2, Configuration 2

Purpose:

SIM directory: SIM_REF_FRAMES RUN directory: SET_test_val/RUN2_ref_frame_ver

The purpose of this test is to demonstrate the relative state and position calculations associated with Reference Frame Model.

Requirements:

By passing this test, the Reference Frame Model partially satisfies the Requirement reframes_5

Procedure:

$x_{A:B}$: $\begin{pmatrix} -4282.53 \\ 4820.22 \\ 1206.12 \end{pmatrix}$

$v_{A:B}$: $\begin{pmatrix} -582.986 \\ 251.806 \\ -290.441 \end{pmatrix}$

$T_{A:B}$: $\begin{pmatrix} 0.9945282144718409 & -0.1036578778650371 & -0.01298749305944376 \\ 0.09724675025241425 & 0.9640168959942612 & -0.2474156296658203 \\ 0.03816674186771286 & 0.2447988329100178 & 0.9688223940542955 \end{pmatrix}$

$w_{A:B}$: $\begin{pmatrix} -98.2663 \\ 73.0249 \\ -34.3877 \end{pmatrix}$

$S_{B:C}$**:**

$x_{A:C}$: $\begin{pmatrix} 1614.85 \\ -1787.66 \\ 4228.86 \end{pmatrix}$

$v_{A:C}$: $\begin{pmatrix} 695.949 \\ 699.888 \\ 638.531 \end{pmatrix}$

$$T_{A:C}: \begin{pmatrix} 0.5617023094980858 & -0.1323969387778994 & 0.8166771492497862 \\ 0.7140778066635471 & 0.5761043624264366 & -0.3977394242765062 \\ -0.4178317861712943 & 0.8065821806830784 & 0.4181408665383421 \end{pmatrix}$$

$$w_{A:C}: \begin{pmatrix} -53.0864 \\ -65.4446 \\ -40.7619 \end{pmatrix}$$

The following relative states will then be calculated:

$S_{B:C}$ $S_{C:B}$

Additionally, the relative positions:

$x_{B:C}$ $x_{C:B}$

will be calculated using the separate, position specific algorithm.

These relative states and positions will then be compared to an independently programmed Matlab relative state and position calculation.

Results:

The following are the results from the independently programmed Matlab relative state solution:

$$x_{B:C}: \begin{pmatrix} 6510.811804678263 \\ -6544.480047115037 \\ 1535.980671550045 \end{pmatrix}$$

$$v_{B:C}: \begin{pmatrix} 114098.2062696465 \\ 73283.19432999958 \\ -166591.9470886627 \end{pmatrix}$$

$$T_{B:C}: \begin{pmatrix} 0.5617461918316968 & -0.2750678528671711 & 0.7802428418650378 \\ 0.6556184084294097 & 0.7232230354607804 & -0.2170551623605903 \\ -0.5045846990259157 & 0.633471481045468 & 0.5866073339219234 \end{pmatrix}$$

$$w_{B:C}: \begin{pmatrix} 49.03187903243332 \\ -61.29672244068031 \\ -116.4326859493806 \end{pmatrix}$$

$$x_{C:B}: \begin{pmatrix} -6656.037735922825 \\ 797.9031862664995 \\ 6529.979956302859 \end{pmatrix}$$

$$v_{C:B}: \begin{pmatrix} 393410.1500972123 \\ -618767.7848463297 \\ 477744.058501485 \end{pmatrix}$$

$$T_{C:B}: \begin{pmatrix} 0.5617461918316968 & 0.6556184084294097 & -0.5045846990259157 \\ -0.2750678528671711 & 0.7232230354607804 & 0.633471481045468 \\ 0.7802428418650378 & -0.2170551623605903 & 0.5866073339219234 \end{pmatrix}$$

$$w_{C:B}: \begin{pmatrix} -46.10636351287094 \\ 131.5750813652961 \\ 16.7387248063523 \end{pmatrix}$$

The following are the results from the Reference Frame Model full relative state calculation:

$$x_{B:C}: \begin{pmatrix} 6510.811804678263 \\ -6544.480047115037 \\ 1535.980671550045 \end{pmatrix}$$

$$v_{B:C}: \begin{pmatrix} 114098.2062696465 \\ 73283.19432999958 \\ -166591.9470886627 \end{pmatrix}$$

$$T_{B:C}: \begin{pmatrix} 0.5617461918316968 & -0.2750678528671711 & 0.7802428418650378 \\ 0.6556184084294097 & 0.7232230354607804 & -0.2170551623605903 \\ -0.5045846990259157 & 0.633471481045468 & 0.5866073339219234 \end{pmatrix}$$

$$w_{B:C}: \begin{pmatrix} 49.03187903243332 \\ -61.29672244068031 \\ -116.4326859493806 \end{pmatrix}$$

$$x_{C:B}: \begin{pmatrix} -6656.037735922825 \\ 797.9031862664995 \\ 6529.979956302859 \end{pmatrix}$$

$$v_{C:B}: \begin{pmatrix} 393410.1500972123 \\ -618767.7848463297 \\ 477744.058501485 \end{pmatrix}$$

$$T_{C:B}: \begin{pmatrix} 0.5617461918316968 & 0.6556184084294097 & -0.5045846990259157 \\ -0.2750678528671711 & 0.7232230354607804 & 0.633471481045468 \\ 0.7802428418650378 & -0.2170551623605903 & 0.5866073339219234 \end{pmatrix}$$

$$w_{C:B}: \begin{pmatrix} -46.10636351287094 \\ 131.5750813652961 \\ 16.7387248063523 \end{pmatrix}$$

The following are the results from the Reference Frame Model position specific calculation:

$$x_{B:C}: \begin{pmatrix} 6510.811804678263 \\ -6544.480047115037 \\ 1535.980671550045 \end{pmatrix}$$

$$x_{C:B}: \begin{pmatrix} -6656.037735922825 \\ 797.9031862664995 \\ 6529.979956302859 \end{pmatrix}$$

All Reference Frame Model produced states are a numerical match to the independently programmed Matlab solution.

Test refframes_6: Relative State Test 3, Configuration 2

Purpose:

SIM directory: SIM_REF_FRAMES RUN directory: SET_test_val/RUN3_ref_frame_ver

The purpose of this test is to demonstrate the relative state and position calculations associated with Reference Frame Model.

Requirements:

By passing this test, the Reference Frame Model partially satisfies the Requirement ref-frames_5

Procedure:

$$x_{A:B}: \begin{pmatrix} -2874.98 \\ -911.135 \\ 5762.09 \end{pmatrix}$$

$$v_{A:B}: \begin{pmatrix} -644.443 \\ -647.618 \\ 679.017 \end{pmatrix}$$

$$T_{A:B}: \begin{pmatrix} -0.1766406884173243 & -0.3584722730823265 & -0.9166764405321235 \\ -0.07753089247582201 & 0.9334947713181074 & -0.3501092295750815 \\ 0.9812171155801128 & 0.009227207212613942 & -0.1926858342995125 \end{pmatrix}$$

$w_{A:B}$: $\begin{pmatrix} -70.9282 \\ 23.6231 \\ 11.9396 \end{pmatrix}$

$S_{B:C}$:

$x_{A:C}$: $\begin{pmatrix} -2564.41 \\ -6134.61 \\ 5822.49 \end{pmatrix}$

$v_{A:C}$: $\begin{pmatrix} -318.074 \\ 119.215 \\ -939.829 \end{pmatrix}$

$T_{A:C}$: $\begin{pmatrix} -0.9774834677306422 & 0.1620640475981063 & -0.1351344322865095 \\ 0.04508295495462142 & 0.786020216558487 & 0.6165547391220908 \\ 0.2061397523182708 & 0.5965798049192318 & -0.7756280931456088 \end{pmatrix}$

$w_{A:C}$: $\begin{pmatrix} -54.4716 \\ -64.7311 \\ -54.3886 \end{pmatrix}$

The following relative states will then be calculated:

$S_{B:C}$ $S_{C:B}$

Additionally, the relative positions:

$x_{B:C}$ $x_{C:B}$

will be calculated using the separate, position specific algorithm.

These relative states and positions will then be compared to an independently programmed Matlab relative state and position calculation.

Results:

The following are the results from the independently programmed Matlab relative state solution:

$x_{B:C}$: $\begin{pmatrix} 1762.244401028797 \\ -4921.311967353402 \\ 244.9002889991167 \end{pmatrix}$

$v_{B:C}$: $\begin{pmatrix} -63392.38080684067 \\ -37153.52608672214 \\ -306790.8805049965 \end{pmatrix}$

$$T_{B:C}: \begin{pmatrix} 0.2384424355108056 & 0.2743829186618951 & -0.9315896193572873 \\ -0.8549111415691051 & 0.5143889358696943 & -0.06731242586516778 \\ 0.4607300130973512 & 0.8124764837220977 & 0.3572251648888851 \end{pmatrix}$$

$$w_{B:C}: \begin{pmatrix} -32.91827495216602 \\ -136.7161862625254 \\ -45.16818828635127 \end{pmatrix}$$

$$x_{C:B}: \begin{pmatrix} 1158.276661310729 \\ 4054.515631124612 \\ 3099.046810448993 \end{pmatrix}$$

$$v_{C:B}: \begin{pmatrix} -19938.73745623053 \\ -105432.3449651296 \\ 144099.0541189501 \end{pmatrix}$$

$$T_{C:B}: \begin{pmatrix} 0.2384424355108056 & -0.8549111415691051 & 0.4607300130973512 \\ 0.2743829186618951 & 0.5143889358696943 & 0.8124764837220977 \\ -0.9315896193572873 & -0.06731242586516778 & 0.3572251648888851 \end{pmatrix}$$

$$w_{C:B}: \begin{pmatrix} -88.2207372355069 \\ 116.0555967214257 \\ -23.73380787662757 \end{pmatrix}$$

The following are the results from the Reference Frame Model full relative state calculation:

$$x_{B:C}: \begin{pmatrix} 1762.244401028797 \\ -4921.311967353402 \\ 244.9002889991167 \end{pmatrix}$$

$$v_{B:C}: \begin{pmatrix} -63392.38080684067 \\ -37153.52608672214 \\ -306790.8805049965 \end{pmatrix}$$

$$T_{B:C}: \begin{pmatrix} 0.2384424355108056 & 0.2743829186618951 & -0.9315896193572873 \\ -0.8549111415691051 & 0.5143889358696943 & -0.06731242586516778 \\ 0.4607300130973512 & 0.8124764837220977 & 0.3572251648888851 \end{pmatrix}$$

$$w_{B:C}: \begin{pmatrix} -32.91827495216602 \\ -136.7161862625254 \\ -45.16818828635127 \end{pmatrix}$$

$$x_{C:B}: \begin{pmatrix} 1158.276661310729 \\ 4054.515631124612 \\ 3099.046810448993 \end{pmatrix}$$

$$v_{C:B}: \begin{pmatrix} -19938.73745623053 \\ -105432.3449651296 \\ 144099.0541189501 \end{pmatrix}$$

$$T_{C:B}: \begin{pmatrix} 0.2384424355108056 & -0.8549111415691051 & 0.4607300130973512 \\ 0.2743829186618951 & 0.5143889358696943 & 0.8124764837220977 \\ -0.9315896193572873 & -0.06731242586516778 & 0.3572251648888851 \end{pmatrix}$$

$$w_{C:B}: \begin{pmatrix} -88.2207372355069 \\ 116.0555967214257 \\ -23.73380787662757 \end{pmatrix}$$

The following are the results from the Reference Frame Model position specific calculation:

$$x_{B:C}: \begin{pmatrix} 1762.244401028797 \\ -4921.311967353402 \\ 244.9002889991167 \end{pmatrix}$$

$$x_{C:B}: \begin{pmatrix} 1158.276661310729 \\ 4054.515631124612 \\ 3099.046810448993 \end{pmatrix}$$

All Reference Frame Model produced states are a numerical match to the independently programmed Matlab solution.

5.3 Metrics

5.3.1 Code Metrics

Table 5.1 presents coarse metrics on the source files that comprise the model.

Table 5.1: Coarse Metrics

	Number of Lines			
File Name	**Blank**	**Comment**	**Code**	**Total**
include/base_ref_frame_manager.hh	38	90	27	155
include/class_declarations.hh	16	25	17	58

Continued on next page

Table 5.1: Coarse Metrics (continued from previous page)

File Name	Number of Lines			
	Blank	**Comment**	**Code**	**Total**
include/ref_frame.hh	74	98	97	269
include/ref_frame_inline.hh	46	87	107	240
include/ref_frame_interface.hh	22	45	14	81
include/ref_frame_items.hh	42	59	46	147
include/ref_frame_items_inline.hh	31	62	69	162
include/ref_frame_links.hh	33	57	28	118
include/ref_frame_manager.hh	49	67	41	157
include/ref_frame_messages.hh	37	79	27	143
include/ref_frame_state.hh	71	100	60	231
include/ref_frame_state_inline.hh	53	86	128	267
include/subscription.hh	86	141	98	325
include/tree_links.hh	106	219	247	572
include/tree_links_iterator.hh	43	120	87	250
src/ref_frame.cc	27	60	48	135
src/ref_frame_compute_relative_state.cc	63	200	155	418
src/ref_frame_items.cc	19	41	46	106
src/ref_frame_manager.cc	77	150	245	472
src/ref_frame_messages.cc	18	27	20	65
src/ref_frame_set_name.cc	30	81	78	189
src/ref_frame_state.cc	113	260	198	571
src/subscription.cc	33	51	98	182
Total	1127	2205	1981	5313

Table 5.2 presents the extended cyclomatic complexity (ECC) of the methods defined in the model.

Table 5.2: Cyclomatic Complexity

Method	File	Line	ECC
jeod::RefFrame::get_name (void)	include/ref_frame_inline.hh	45	1

Continued on next page

Table 5.2: Cyclomatic Complexity (continued)

Method	File	Line	ECC
jeod::RefFrame::set_owner (RefFrameOwner * new_ owner)	include/ref_frame_inline.hh	58	1
jeod::RefFrame::get_owner (void)	include/ref_frame_inline.hh	71	1
jeod::RefFrame::get_parent (void)	include/ref_frame_inline.hh	84	1
jeod::RefFrame::get_root (void)	include/ref_frame_inline.hh	97	1
jeod::RefFrame::set_ timestamp (double time)	include/ref_frame_inline.hh	110	1
jeod::RefFrame::timestamp (void)	include/ref_frame_inline.hh	123	1
jeod::RefFrame::make_root (void)	include/ref_frame_inline.hh	136	1
jeod::RefFrame::add_child (RefFrame & frame)	include/ref_frame_inline.hh	149	1
jeod::RefFrame::remove_from_ parent (void)	include/ref_frame_inline.hh	163	1
jeod::RefFrame::find_last_ common_index (const Ref Frame & frame)	include/ref_frame_inline.hh	176	1
jeod::RefFrame::find_last_ common_node (const Ref Frame & frame)	include/ref_frame_inline.hh	193	2
jeod::RefFrame::is_progeny_of (const RefFrame & frame)	include/ref_frame_inline.hh	217	1
jeod::RefFrameOwner::Ref FrameOwner ()	include/ref_frame_interface.hh	52	1
jeod::RefFrameOwner::~Ref FrameOwner ()	include/ref_frame_interface.hh	57	1
jeod::RefFrameOwner::note_ frame_status_change (Ref Frame * frame JEOD_UNU SED)	include/ref_frame_interface.hh	63	1
jeod::RefFrameItems::get (void)	include/ref_frame_items_ inline.hh	38	1

Continued on next page

Table 5.2: Cyclomatic Complexity (continued)

Method	File	Line	ECC
jeod::RefFrameItems::contains (RefFrameItems::Items test_items)	include/ref_frame_items_inline.hh	51	1
jeod::RefFrameItems::equals (RefFrameItems::Items test_items)	include/ref_frame_items_inline.hh	67	1
jeod::RefFrameItems::is_empty (void)	include/ref_frame_items_inline.hh	81	1
jeod::RefFrameItems::is_full (void)	include/ref_frame_items_inline.hh	94	1
jeod::RefFrameItems::set (Ref FrameItems::Items new_value)	include/ref_frame_items_inline.hh	107	1
jeod::RefFrameItems::add (RefFrameItems::Items new_items)	include/ref_frame_items_inline.hh	121	1
jeod::RefFrameItems::remove (RefFrameItems::Items old_items)	include/ref_frame_items_inline.hh	137	1
jeod::RefFrameLinks::~Ref FrameLinks (void)	include/ref_frame_links.hh	76	1
jeod::RefFrameTrans::Ref FrameTrans (void)	include/ref_frame_state_inline.hh	44	1
jeod::RefFrameTrans::Ref FrameTrans (const Ref FrameTrans & source)	include/ref_frame_state_inline.hh	55	1
jeod::RefFrameTrans::~Ref FrameTrans (void)	include/ref_frame_state_inline.hh	67	1
jeod::RefFrameTrans:: initialize (void)	include/ref_frame_state_inline.hh	78	1
jeod::RefFrameTrans::copy (const RefFrameTrans & source)	include/ref_frame_state_inline.hh	90	1
jeod::RefFrameRot::RefFrame Rot (void)	include/ref_frame_state_inline.hh	104	1
jeod::RefFrameRot::RefFrame Rot (const RefFrameRot & source)	include/ref_frame_state_inline.hh	115	1

Continued on next page

Table 5.2: Cyclomatic Complexity (continued)

Method	File	Line	ECC
jeod::RefFrameRot::~Ref FrameRot (void)	include/ref_frame_state_ inline.hh	127	1
jeod::RefFrameRot::initialize (void)	include/ref_frame_state_ inline.hh	138	1
jeod::RefFrameRot::copy (const RefFrameRot & source)	include/ref_frame_state_ inline.hh	153	1
jeod::RefFrameRot::compute_ transformation (void)	include/ref_frame_state_ inline.hh	169	1
jeod::RefFrameRot::compute_ quaternion (void)	include/ref_frame_state_ inline.hh	180	1
jeod::RefFrameRot::compute_ ang_vel_unit (void)	include/ref_frame_state_ inline.hh	191	2
jeod::RefFrameRot::compute_ ang_vel_products (void)	include/ref_frame_state_ inline.hh	210	1
jeod::RefFrameState::~Ref FrameState (void)	include/ref_frame_state_ inline.hh	223	1
jeod::RefFrameState::initialize (void)	include/ref_frame_state_ inline.hh	234	1
jeod::RefFrameState::copy (const RefFrameState & source)	include/ref_frame_state_ inline.hh	246	1
jeod::ActivateInterface:: ActivateInterface ()	include/subscription.hh	50	1
jeod::ActivateInterface::~ ActivateInterface ()	include/subscription.hh	55	1
jeod::SubscribeInterface:: SubscribeInterface ()	include/subscription.hh	85	1
jeod::SubscribeInterface::~ SubscribeInterface ()	include/subscription.hh	90	1
jeod::Subscription:: Subscription (void)	include/subscription.hh	223	1
jeod::Subscription:: Subscription (Mode init_ mode)	include/subscription.hh	238	1
jeod::Subscription::~ Subscription (void)	include/subscription.hh	254	1

Continued on next page

Table 5.2: Cyclomatic Complexity (continued)

Method	File	Line	ECC
jeod::Subscription::is_active (void)	include/subscription.hh	265	1
jeod::Subscription:: subscriptions (void)	include/subscription.hh	279	1
jeod::Subscription::set_ subscription_mode (Mode value)	include/subscription.hh	292	1
jeod::Subscription::get_ subscription_mode (void)	include/subscription.hh	304	1
jeod::Links::(std::TreeLinks (Container & container_in, unsigned int path_size)	include/tree_links.hh	78	1
jeod::Links::child_head ()	include/tree_links.hh	112	1
jeod::Links::child_tail ()	include/tree_links.hh	120	1
jeod::Links::is_atomic ()	include/tree_links.hh	129	1
jeod::Links::has_children ()	include/tree_links.hh	138	1
jeod::Links::is_root ()	include/tree_links.hh	148	1
jeod::Links::container ()	include/tree_links.hh	158	1
jeod::Links::container ()	include/tree_links.hh	167	1
jeod::Links::links_parent ()	include/tree_links.hh	177	1
jeod::Links::links_parent ()	include/tree_links.hh	186	1
jeod::Links::parent ()	include/tree_links.hh	196	2
jeod::Links::parent ()	include/tree_links.hh	205	2
jeod::Links::links_root ()	include/tree_links.hh	215	1
jeod::Links::links_root ()	include/tree_links.hh	224	1
jeod::Links::root ()	include/tree_links.hh	234	1
jeod::Links::root ()	include/tree_links.hh	243	1
jeod::Links::path_length ()	include/tree_links.hh	253	1
jeod::Links::std::find_path_ index (const Links& link)	include/tree_links.hh	262	1
jeod::Links::nth_from_root (unsigned int index)	include/tree_links.hh	272	2
jeod::Links::nth_from_root (unsigned int index)	include/tree_links.hh	283	2
jeod::Links::MessageHandler:: make_root ()	include/tree_links.hh	295	2

Continued on next page

Table 5.2: Cyclomatic Complexity (continued)

Method	File	Line	ECC
jeod::Links::MessageHandler:: attach (Links & new_ parent)	include/tree_links.hh	313	3
jeod::Links::detach ()	include/tree_links.hh	346	1
jeod::Links::reattach (Links & new_parent)	include/tree_links.hh	359	1
jeod::Links::is_progeny_of (const Links & target)	include/tree_links.hh	378	3
jeod::Links::std::find_last_ common_index (const Links & target)	include/tree_links.hh	398	6
jeod::Links::find_last_ common_node (const Links & target)	include/tree_links.hh	441	2
jeod::Links::std::construct_ path_to_node ()	include/tree_links.hh	458	3
jeod::Links::attach_internal (Links & new_parent)	include/tree_links.hh	489	1
jeod::Links::std::detach_ internal ()	include/tree_links.hh	503	2
jeod::Links::std::set_path_size (unsigned int new_size)	include/tree_links.hh	520	1
jeod::TreeLinksRange::Tree LinksRange (T begin_in, U end_in)	include/tree_links_iterator.hh	89	1
jeod::TreeLinksRange::begin ()	include/tree_links_iterator.hh	104	1
jeod::TreeLinksRange::end ()	include/tree_links_iterator.hh	109	1
jeod::TreeLinksAscendRange:: TreeLinksAscendRange (Links & links)	include/tree_links_iterator.hh	142	1
jeod::TreeLinksAscendRange:: TreeLinksAscendRange (Links & links, unsigned int start_index, unsigned int end_index = 0)	include/tree_links_iterator.hh	155	1

Continued on next page

Table 5.2: Cyclomatic Complexity (continued)

Method	File	Line	ECC
jeod::TreeLinksDescent Range::TreeLinksDescent Range (Links & links, unsigned int start_index = 0)	include/tree_links_iterator.hh	197	1
jeod::TreeLinksChildren Range::TreeLinksChildren Range (Links & links)	include/tree_links_iterator.hh	229	1
jeod::RefFrame::RefFrame (void)	src/ref_frame.cc	46	1
jeod::RefFrame::~RefFrame ()	src/ref_frame.cc	61	2
jeod::RefFrame::set_active_ status (bool value)	src/ref_frame.cc	76	2
jeod::RefFrame::transplant_ node (RefFrame & new_ parent)	src/ref_frame.cc	96	1
jeod::RefFrame::reset_parent (RefFrame & new_parent)	src/ref_frame.cc	118	1
jeod::RefFrame::compute_ relative_state (const Ref Frame & wrt_frame, Ref FrameState & rel_state)	src/ref_frame_compute_ relative_state.cc	45	6
jeod::RefFrame::compute_ relative_state (const Ref Frame & wrt_frame, bool reverse_sense, RefFrame State & rel_state)	src/ref_frame_compute_ relative_state.cc	138	2
jeod::RefFrame::compute_ state_wrt_pred (const Ref Frame & pred_frame, Ref FrameState & rel_state)	src/ref_frame_compute_ relative_state.cc	191	2
jeod::RefFrame::compute_ state_wrt_pred (unsigned int pred_frame_index, Ref FrameState & rel_state)	src/ref_frame_compute_ relative_state.cc	225	2
jeod::RefFrame::compute_ pred_rel_state (const Ref Frame & pred_frame, Ref FrameState & rel_state)	src/ref_frame_compute_ relative_state.cc	267	2

Continued on next page

Table 5.2: Cyclomatic Complexity (continued)

Method	File	Line	ECC
jeod::RefFrame::compute_pred_rel_state (unsigned int pred_frame_index, RefFrameState & rel_state)	src/ref_frame_compute_relative_state.cc	301	2
jeod::RefFrame::compute_position_from (const RefFrame& in_frame, double rel_pos[3])	src/ref_frame_compute_relative_state.cc	345	6
jeod::RefFrameItems::to_string (Items test_items)	src/ref_frame_items.cc	33	17
jeod::RefFrameItems::RefFrameItems (void)	src/ref_frame_items.cc	67	1
jeod::RefFrameItems::RefFrameItems (Items new_value)	src/ref_frame_items.cc	77	1
jeod::RefFrameItems::to_string (void)	src/ref_frame_items.cc	88	1
jeod::RefFrameManager::RefFrameManager (void)	src/ref_frame_manager.cc	42	1
jeod::RefFrameManager::~RefFrameManager (void)	src/ref_frame_manager.cc	58	1
jeod::RefFrameManager::add_ref_frame (RefFrame & ref_frame)	src/ref_frame_manager.cc	73	5
jeod::RefFrameManager::remove_ref_frame (RefFrame & ref_frame)	src/ref_frame_manager.cc	117	2
jeod::RefFrameManager::find_ref_frame (const char * name)	src/ref_frame_manager.cc	140	3
jeod::RefFrameManager::find_ref_frame (const char * prefix, const char * suffix)	src/ref_frame_manager.cc	168	5
jeod::RefFrameManager::check_ref_frame_ownership (void)	src/ref_frame_manager.cc	202	4
jeod::RefFrameManager::reset_tree_root_node (void)	src/ref_frame_manager.cc	225	1

Continued on next page

Table 5.2: Cyclomatic Complexity (continued)

Method	File	Line	ECC
jeod::RefFrameManager::add_frame_to_tree (RefFrame & ref_frame, RefFrame * parent)	src/ref_frame_manager.cc	236	2
jeod::RefFrameManager::subscribe_to_frame (const char * frame_name)	src/ref_frame_manager.cc	266	3
jeod::RefFrameManager::subscribe_to_frame (Ref Frame & frame)	src/ref_frame_manager.cc	301	1
jeod::RefFrameManager::unsubscribe_to_frame (const char * frame_name)	src/ref_frame_manager.cc	317	3
jeod::RefFrameManager::unsubscribe_to_frame (Ref Frame & frame)	src/ref_frame_manager.cc	351	2
jeod::RefFrameManager::frame_is_subscribed (const char * frame_name)	src/ref_frame_manager.cc	375	3
jeod::RefFrameManager::frame_is_subscribed (Ref Frame & frame)	src/ref_frame_manager.cc	406	1
jeod::RefFrameManager::validate_name (const char * file, unsigned int line, const char * variable_value, const char * variable_type, const char * variable_name)	src/ref_frame_manager.cc	424	3
jeod::RefFrame::set_name (const char * name_item1)	src/ref_frame_set_name.cc	37	1
jeod::RefFrame::set_name (const char * name_item1, const char * name_item2)	src/ref_frame_set_name.cc	49	1
jeod::RefFrame::set_name (const char * name_item1, const char * name_item2, const char * name_item3)	src/ref_frame_set_name.cc	64	1

Continued on next page

Table 5.2: Cyclomatic Complexity (continued)

Method	File	Line	ECC
jeod::RefFrame::set_name (const char * name_item1, const char * name_item2, const char * name_item3, const char * name_item4)	src/ref_frame_set_name.cc	84	1
jeod::RefFrame::set_name (const char * name_item1, const char * name_item2, const char * name_item3, const char * name_item4, const char * name_item5)	src/ref_frame_set_name.cc	106	1
jeod::RefFrame::set_name (const char * name_item1, const char * name_item2, const char * name_item3, const char * name_item4, const char * name_item5, const char * name_item6)	src/ref_frame_set_name.cc	130	1
jeod::RefFrame::set_name (const char * name_item1, const char * name_item2, const char * name_item3, const char * name_item4, const char * name_item5, const char * name_item6, const char * name_item7)	src/ref_frame_set_name.cc	156	1
jeod::RefFrameTrans:: operator = (const Ref FrameTrans & source)	src/ref_frame_state.cc	119	2
jeod::RefFrameRot::operator = (const RefFrameRot & source)	src/ref_frame_state.cc	134	2
jeod::RefFrameState::Ref FrameState (void)	src/ref_frame_state.cc	151	1
jeod::RefFrameState::Ref FrameState (const Ref FrameState & source)	src/ref_frame_state.cc	161	1
jeod::RefFrameState::operator = (const RefFrameState & source)	src/ref_frame_state.cc	172	2

Continued on next page

Table 5.2: Cyclomatic Complexity (continued)

Method	File	Line	ECC
jeod::RefFrameState::negate (const RefFrameState & source)	src/ref_frame_state.cc	188	3
jeod::RefFrameState::incr_left (const RefFrameState & s_ab)	src/ref_frame_state.cc	247	5
jeod::RefFrameState::incr_right (const RefFrameState & s_bc)	src/ref_frame_state.cc	331	3
jeod::RefFrameState::decr_left (const RefFrameState & s_ab)	src/ref_frame_state.cc	420	3
jeod::RefFrameState::decr_right (const RefFrameState & s_bc)	src/ref_frame_state.cc	490	5
jeod::Subscription::activate (void)	src/subscription.cc	32	4
jeod::Subscription::deactivate (void)	src/subscription.cc	62	4
jeod::Subscription::subscribe (void)	src/subscription.cc	92	4
jeod::Subscription::unsubscribe (void)	src/subscription.cc	124	6
jeod::Subscription::set_active_status (bool value)	src/subscription.cc	164	1

5.4 Requirements Traceability

Table 5.3: Requirements Traceability

Requirement	Inspection and Testing
reframes_1 - Top-level Requirements	Inspection reframes_1
reframes_2 - Reference Frame Representation Requirement	Inspection reframes_2
reframes_3 - Reference Frame State Requirement	Inspection reframes_3
reframes_4 - Reference Frame Tree Structure	Inspection reframes_4
reframes_8 - Reference Frame Manager Utility	Inspection reframes_7

Table 5.3: Requirements Traceability (continued)

Requirement	**Inspection and Testing**
8.1 - Searchable Registry	Inspection refframes_7
8.2 - Tree Building Functionality	Inspection refframes_7
8.3 - Subscription Mechanism Access	Inspection refframes_7
refframes_5 - Reference Frame Relative State Calculation	Test refframes_1 Test refframes_2 Test refframes_3 Test refframes_4 Test refframes_5 Test refframes_6
refframes_6 - Reference Frame Active/Inactive Functionality	Inspection refframes_5
refframes_7 - Reference Frame Subscription Functionality	Inspection refframes_6

Bibliography

[1] Generated by doxygen. *Reference Frames Reference Manual*. NASA, Johnson Space Center, Software, Robotics & Simulation Division, Simulation and Graphics Branch, 2101 NASA Parkway, Houston, Texas, 77058, July 2022.

[2] Hammen, D. Dynamic Body Model. Technical Report JSC-61777-dynamics/dyn_body, NASA, Johnson Space Center, Houston, Texas, July 2022.

[3] Hammen, D. Body Action Model. Technical Report JSC-61777-dynamics/body_action, NASA, Johnson Space Center, Houston, Texas, July 2022.

[4] Hammen, D. Quaternion Model. Technical Report JSC-61777-utils/quaternion, NASA, Johnson Space Center, Houston, Texas, July 2022.

[5] Hammen, D. Dynamics Manager Model. Technical Report JSC-61777-dynamics/dyn_manager, NASA, Johnson Space Center, Houston, Texas, July 2022.

[6] Jackson, A., Thebeau, C. JSC Engineering Orbital Dynamics. Technical Report JSC-61777-docs, NASA, Johnson Space Center, Houston, Texas, July 2022.

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

[8] Shelton, R. Message Handler Model. Technical Report JSC-61777-utils/message, NASA, Johnson Space Center, Houston, Texas, July 2022.

[9] Thompson, B. Ephemerides Model. Technical Report JSC-61777-environment/ephemerides, NASA, Johnson Space Center, Houston, Texas, July 2022.

[10] Turner, G. Derived State Model. Technical Report JSC-61777-dynamics/derived_state, NASA, Johnson Space Center, Houston, Texas, July 2022.

[11] Turner, G. Time Model. Technical Report JSC-61777-environment/time, NASA, Johnson Space Center, Houston, Texas, July 2022.

[12] Vallado, D.A. *Fundamentals of Astrodynamics and Applications*. McGraw-Hill, New York, 1997.