Technical Report # TR–93–06

A Formal Specification of the RSDIMU Inertial Navigation System

Steven Zeil, Aileen Biser, Linghan Cai, Hong Huang,
Tijen Ireland, Brian Mitchell, and George Walker

Old Dominion University
Department of Computer Science
Norfolk, VA 23529-0162
U.S.A.

March 12, 1993
Updated: April 7, 1993

This work was supported by grant NAG-1-439 from the NASA Langley Research Center.

Abstract

This document presents a formal specification for the RSDIMU, a component of an aircraft inertial navigation system. The specification is written in the Z language.

The RSDIMU system consists of a set of eight acceleration sensors, mounted upon the four upright faces of a square based pyramid (i.e., a semi-octohedron), two sensors per face. The purpose of this sensor array is to provide a measure of the current acceleration of the aircraft within which it is mounted. The eight sensors provide redundancy during the estimate of the 3-dimensional acceleration. This redundancy permits valid accleration estimates in the face of sensor failures and aids in reducing error due to noise in the sensor readings.
Keywords: Formal Specifications, Z, RSDIMU

Contents

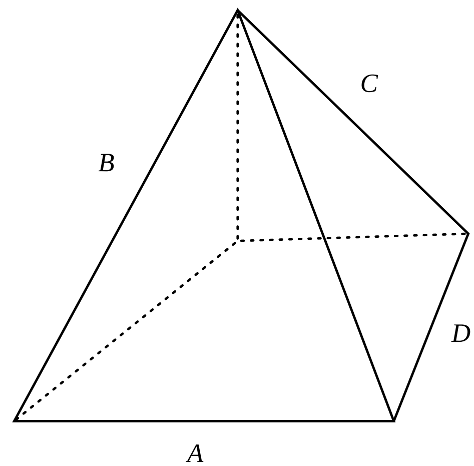

Figure 1: The RSDIMU Instrument Package

1 Introduction

1.1 The RSDIMU

This document presents a formal specification for the RSDIMU navigation sensor system. The specification is written in the Z language.

The RSDIMU problem was previously isolated as a subject for studies in the effectiveness of software redundancy for fault tolerance [3, 4]. A requirements document [1] was developed jointly by staff of the Research Triangle Institute and of Charles River Analytics.

The RSDIMU (**R**edundant **S**trapped-**D**own **I**nertial **M**easurement **U**nit) system consists of a set of eight acceleration sensors, mounted upon the four upright faces of a square based pyramid (i.e., half of a regular octahedron), two sensors per face. Figures 1 and 2 illustrate this structure. The purpose of this sensor array is to provide a measure of the current acceleration of the aircraft within which it is mounted. The eight sensors provide redundancy during the estimate of the 3-dimensional acceleration. This redundancy permits valid acceleration estimates in the face of sensor failures and aids in reducing error due to noise in the sensor readings. The redundancy management software for the RSDIMU is charged with the following tasks:

- Calibration of the accelerometers with the aircraft at rest (subject only to gravitational acceleration),
- Analysis of noise in the accelerometer readings,
- Detection of failed sensors based upon
  - excessive noise in readings from an individual sensor, or
  - inconsistency of an individual sensor's readings as compared to readings from the others,
- Estimation of the aircraft acceleration using those calibrated sensors deemed to be operational.

This specification covers the description of the RSDIMU hardware interface, the pre-flight calibration, and a subsequent single (instantaneous) estimate of vehicle acceleration while in-flight. More realistically, we would expect the RSDIMU to be included within a larger navigation system that would perform a continuous series of such estimates, integrating them over time to determine the

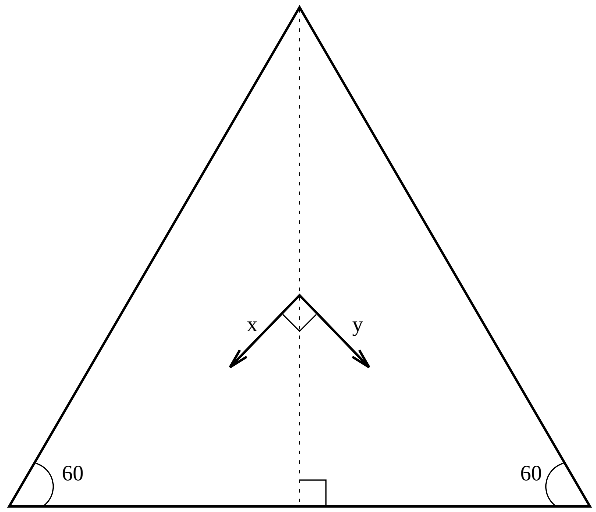

Figure 2: Sensor Mounting on RSDIMU Faces

position and velocity of the aircraft. In [1], however, the validation process calls for only a single estimate.

1.2 History of This Specification

The specification presented in this document was developed by the instructor and students of the Fall 1992 class in "Formal Methods in Software Engineering" at the Old Dominion University Department of Computer Science. After approximately 7 weeks instruction in notations for writing formal specifications, with emphasis on the Z language [2], the instructor (Steven Zeil) presented the students with the general framework for the Z specification. This framework corresponds roughly to Sections 2 and 3 of the following document. Responsibility for the remainder of the requirements specification [1] was then distributed among the students. In particular, students were asked to specify the process of calibration and sensor failure detection, the process of estimating vehicle acceleration from the calibrated sensors, and the hardware/software subsystem for the RSDIMU display panel. In the course of developing these specifications, the students suggested various changes to the instructor's original specification of the physical components of the sensor array (Section 3). The resulting specifications have been collected by the instructor to form this document.

The organization of this document follows a roughly bottom-up presentation of the RSDIMU. Section 2 presents the "background" mathematics upon which the RSDIMU is based — vectors and transformations in 3-space. Section 3 presents the physical construction of the RSDIMU sensor array. Section 4 describes the calibration and failure detection procedures. Section 5 describes the process of estimating the current acceleration of the vehicle. Finally, Section 6 specifies the RSDIMU output display panel.

1.3 Conventions

Most of the notation used here is standard Z. This has been augmented in a few instances with standard linear algebraic conventions (such as the ability to write vectors and matrices in the usual rectangular forms).

In addition, certain lexical conventions are employed in selecting names. Names of individual objects or variables begin with lower-case letters, although upper case letters are used within multiword names to indicate the start of each distinct word. Schema names and data type names each begin with upper-case letters. The names of schemas that indicate state (rather than state transitions) are generally singular noun phrases. Data type names, on the other hand, are plural noun phrases. These two often occur in related pairs. For example, "*Sensor*" is the name of a schema describing the state of an arbitrary linear acceleration sensor. "*Sensors*" is the name of the data type comprised of all possible objects that satisfy the *Sensor* schema.

In the original requirements document [1], a number of names are given for objects that are required to appear as types, variables, or constants in an acceptable implementation. To maintain a clear link to that document, we have attempted to use those names whenever possible, even in cases where we believe that clearer names would have contributed to easier understandability. For example, the name *linstd* denotes the maximum acceptable value for the standard deviation of accelerometer readings from any sensor. While the "std" part of the name clearly indicates standard deviation, and the "lin" is a convention used consistently to indicate **in**put from a **l**inear accelerometer, nothing in this name suggests that it represents a maximum threshold rather than an actual standard deviation value for some sensor. We might have preferred a name max_linstd or $max\_operational\_lin\_std$, but have opted to keep the name "as is" to maintain compatibility with [1].

Some consistent changes are made to the names taken from [1]. The requirements document spells out the need to maintain separate variables for many input and output quantities. For example, an array of sensor failure information occurs in [1] as *linfailin* and *linfailout*. Because Z is quite adept at distinguishing input and output quantities, we have consistently removed the "in" and "out" designations from variable names, preferring instead to employ the Z decorations. For example, we have *linfail* for the failure information prior to any state change, and $linfail'$ for the updated failure information after a state change.

2 Coordinate Systems

2.1 Basic Definitions

This section sets up some basic concepts about coordinate systems and transformations of point/vector coordinates from one system to another.

To begin with, define 3-dimensional vectors and matrices, indexed by the typical direction names x, y, z.

$$DirectionNames == \{x, y, z\}$$

$$Vectors3D == DirectionNames \rightarrow \Re$$

$$TransformMatrices == (DirectionNames \times DirectionNames) \rightarrow \Re$$

I will assume that the conventional linear algebraic operations have been defined on these types, and that the conventional vector/matrix tabular forms are understood.

Some convenient constants are

$$\begin{array}{|l}
identity : TransformMatrices \\
zero3D : Vectors3D \\
\hline
identity = \{\ d : DirectionNames \bullet (d, d) \mapsto 1.0\ \} \\
\qquad \cup \{\ d1, d2 : DirectionNames \mid d1 \neq d2 \bullet (d1, d2) \mapsto 0.0\ \} \\
zero3D = \{\ d : DirectionNames \bullet d \mapsto 0.0\ \}
\end{array}$$

or, alternatively

$$
\begin{array}{|l}
identity : TransformMatrices \\
zero3D : Vectors3D \\
\hline
identity = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\
zero3D = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}
\end{array}
$$

In defining directions and points, we have two choices. Both are conventionally written as a 3-tuple of $\Re$, in other words, a $Vectors3D$. That, however, assumes that the coordinate system is known from context. In this application, we are dealing with so many different coordinate systems, a simple vector of three numbers would be easily misinterpreted. So we will say that directions and points must carry their appropriate coordinate system with them:

$[Frames]$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\ Point\ }} \\
coord : Vectors3D \\
x, y, z : \Re \\
system : Frames \\
\hline
x = coord(x) \\
y = coord(y) \\
z = coord(z) \\
\hline
\end{array}
$$

As a convenience, the coordinates of a point p can be accessed individually $(p.x, p.y, p.z)$ or as an entire coordinate vector $(p.coord)$.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\ Direction\ }} \\
Point \\
\hline
\sqrt{coord(x)^2 + coord(y)^2 + coord(z)^2} = 1.0 \\
\hline
\end{array}
$$

We will want to use points and directions as types, so next we define appropriate sets to serve as types:

$Points == \{Point\}$

$Directions == \{Direction\}$

2.2 Transformations

A Transformation describes how to move from one coordinate system (frame) to another:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\underline{\ Transformation\ }} \\
rotation : TransformMatrices \\
translation : Vectors3D \\
\hline
\end{array}
$$

$Transformations == \{Transformation\}$

Using these, we can define a function transforming vectors from one coordinate system to another:

$$
\begin{array}{|l}
transform : (\,Vectors3D \times Transformations) \rightarrow Vectors3D \\
\hline
transform = \lambda\, v : Vectors3D;\ t : Transformations \bullet \\
\qquad t.rotation * (v + t.translation)
\end{array}
$$

A frame identifies a coordinate system. Most frames are defined relative to other frames. Transformations are transitive. If we can transform from frame A to frame B, and can transform from B to C, then we can also transform directly from A to C. Because of this, if we are describing a world containing many frames, we need not give transformations between each pair of frames, but only those frames that are most closely and simply related to one another.

A world, then is a collection of related frames:

$$
\begin{array}{|l}
Worlds : \mathbb{P}(Frames \times Frames) \nrightarrow Transformations \\
\hline
\forall\, w : Worlds;\ f : Frames \bullet \\
\qquad w(f,f).rotation = identity \wedge w(f,f).translation = zero3D
\end{array}
$$

What if we want to transform coordinates between two frames not directly related within our world? If we can find a chain of related frames, we can come up with the composite transformation information:

$$
\begin{array}{|l}
translation\_between : (Frames \times Frames \times Worlds) \nrightarrow Vectors3D \\
\hline
translation\_between = \\
\qquad \{\, f1, f2 : Frames;\ w : Worlds;\ t : Transformations \mid \\
\qquad\qquad (f1,f2) \mapsto t \in w \bullet (f1,f2,w) \mapsto t.translation \,\} \\
\qquad \cup \{\, f1, f2, f3 : Frames;\ w : Worlds;\ t : Transformations \mid \\
\qquad\qquad (f1,f2) \mapsto t \in w \bullet (f1,f3,w) \mapsto t.translation + translation\_between(f2,f3,w) \,\}
\end{array}
$$

$$
\begin{array}{|l}
rotation\_between : (Frames \times Frames \times Worlds) \nrightarrow TransformMatrices \\
\hline
rotation\_between = \\
\qquad \{\, f1, f2 : Frames;\ w : Worlds;\ t : TransformMatrices \mid \\
\qquad\qquad (f1,f2) \mapsto t \in w \bullet (f1,f2,w) \mapsto t.rotation \,\} \\
\qquad \cup \{\, f1, f2, f3 : Frames;\ w : Worlds;\ t : TransformMatrices \mid \\
\qquad\qquad (f1,f2) \mapsto t \in w \bullet (f1,f3,w) \mapsto t.rotation * rotation\_between(f2,f3,w) \,\}
\end{array}
$$

Using these, we can define a function transforming points from one coordinate system to another within a world:

$$
\begin{array}{|l}
\hline
\textit{DoTransform} \\
p? : Points \\
w? : Worlds \\
from?, to? : Frames \\
p! : Points \\
t : Transformations \\
\hline
t.rotation = rotation\_between(from?, to?, w?) \\
t.translation = translation\_between(from?, to?, w?) \\
p!.coord = transform(p?.coord, t) \\
p!.system = to? \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
transform : (Points \times Frames \times Worlds) \rightarrow\!\!\!\!\!\!+ \; Points \\
\hline
transform = \{\ DoTransform \bullet (p?, from?, to?, w?) \mapsto p!\ \}
\end{array}
$$

2.3 Special-Case Transformations

Transformations between frames may be computed in many ways. This section is devoted to various special case approaches to obtaining transformations. In particular, we consider the designation of a rotation by yaw, pitch and roll angles, the designation of a general rotation through very small angles, and transformations among the planar faces of a semi-octahedron.

2.3.1 Yaw, Pitch, and Roll

If both frames are orthogonal, the rotation is often specified by three angles called the "yaw", "pitch", and "roll". The yaw angle, often denoted by ψ, is a rotation about the z-axis and yields a transform

$$
\begin{array}{|l}
yaw : Angles \rightarrow TransformMatrices \\
\hline
yaw = \left\{ \psi : Angles \bullet \psi \mapsto \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \right\}
\end{array}
$$

The pitch is a subsequent rotation about the resulting Y axis (after applying the yaw).

$$
\begin{array}{|l}
pitch : Angles \rightarrow TransformMatrices \\
\hline
pitch = \left\{ \theta : Angles \bullet \theta \mapsto \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \right\}
\end{array}
$$

The roll is a subsequent rotation about the resulting X axis (after applying the yaw and pitch).

$$
\begin{array}{|l}
roll : Angles \rightarrow TransformMatrices \\
\hline
roll = \left\{ \phi : Angles \bullet \phi \mapsto \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \right\}
\end{array}
$$

The composite transformation is then defined as

$$
\begin{array}{|l}
yaw\_pitch\_roll : (Angles \times Angles \times Angles) \rightarrow TransformMatrices \\
\hline
yaw\_pitch\_roll(\psi, \theta, \phi) = yaw(\psi) * pitch(\theta)) * roll(\phi)
\end{array}
$$

We can also define its inverse:

$$
\begin{array}{|l}
roll\_pitch\_yaw : (Angles \times Angles \times Angles) \rightarrow TransformMatrices \\
\hline
roll\_pitch\_yaw(\psi, \theta, \phi) = roll(-\phi) * pitch(-\theta) * yaw(-\psi)
\end{array}
$$

2.3.2 Misalignment

Misalignment describes a small rotation into a (usually) non-orthogonal coordinate system. For each of the three axes in the "ideal" coordinate system, the corresponding axis in the misalignment system is described by a pair of small rotations around the other two ideal axes. These rotations are "small" in the sense that they are rotations through an angle θ small enough that θ and $\sin\theta$ are approximately equal (when θ is measured in radians).

Each misalignment angle is labeled with a pair of axes names.

$$MisalignmentNames == \{xy, xz, yx, yz, zx, zy\}$$

$$MisalignmentAngles == MisalignmentNames \rightarrow Angles$$

The first name is the axis described by the misalignment angle. The second is the axis about which the rotation occurs. For example, θ_{xy} describes the rotation of the x axis around the y axis.

The misalignment angles can be used to obtain a transform matrix as follows

$$toMisaligned : MisalignmentAngles \rightarrow TransformMatrices$$

$$toMisaligned = \left\{ \theta : MisalignmentAngles \bullet \begin{bmatrix} 1 & \theta(xz) & -\theta(xy) \\ -\theta(yz) & 1 & \theta(yx) \\ \theta(zy) & -\theta(zx) & 1 \end{bmatrix} \right\}$$

The inverse is

$$fromMisaligned : MisalignmentAngles \rightarrow TransformMatrices$$

$$fromMisaligned = \left\{ \theta : MisalignmentAngles \bullet \begin{bmatrix} 1 & -\theta(xz) & \theta(xy) \\ \theta(yz) & 1 & -\theta(yx) \\ -\theta(zy) & \theta(zx) & 1 \end{bmatrix} \right\}$$

2.3.3 Pyramids

The following definitions describe the transformation from a single frame I into a series of four frames whose $x - y$ planes form a semi-octahedron with its base centered on the origin of the I frame.[1]

It's not exciting, but it is necessary.

Define the following family of vectors:

$$StoI : (FaceNames \times DirectionNames) \twoheadrightarrow Vectors3D$$

$$StoI =$$

$$\left\{ \begin{array}{lll}
(A,x) \mapsto \frac{1}{2\sqrt{3}} \begin{bmatrix} \sqrt{3}+1 \\ -\sqrt{3}+1 \\ -2 \end{bmatrix} & (A,y) \mapsto \frac{1}{2\sqrt{3}} \begin{bmatrix} -\sqrt{3}+1 \\ \sqrt{3}+1 \\ -2 \end{bmatrix} & (A,z) \mapsto \frac{1}{\sqrt{3}} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \\
(B,x) \mapsto \frac{1}{2\sqrt{3}} \begin{bmatrix} -\sqrt{3}+1 \\ -\sqrt{3}-1 \\ -2 \end{bmatrix} & (B,y) \mapsto \frac{1}{2\sqrt{3}} \begin{bmatrix} \sqrt{3}+1 \\ \sqrt{3}-1 \\ -2 \end{bmatrix} & (B,z) \mapsto \frac{1}{\sqrt{3}} \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} \\
(C,x) \mapsto \frac{1}{2\sqrt{3}} \begin{bmatrix} -\sqrt{3}-1 \\ \sqrt{3}-1 \\ -2 \end{bmatrix} & (C,y) \mapsto \frac{1}{2\sqrt{3}} \begin{bmatrix} \sqrt{3}-1 \\ -\sqrt{3}-1 \\ -2 \end{bmatrix} & (C,z) \mapsto \frac{1}{\sqrt{3}} \begin{bmatrix} -1 \\ -1 \\ 1 \end{bmatrix} \\
(D,x) \mapsto \frac{1}{2\sqrt{3}} \begin{bmatrix} \sqrt{3}-1 \\ \sqrt{3}+1 \\ -2 \end{bmatrix} & (D,y) \mapsto \frac{1}{2\sqrt{3}} \begin{bmatrix} \sqrt{3}-1 \\ -\sqrt{3}+1 \\ -2 \end{bmatrix} & (D,z) \mapsto \frac{1}{\sqrt{3}} \begin{bmatrix} -1 \\ 1 \\ 1 \end{bmatrix}
\end{array} \right\}$$

Then we can define the following transformation matrices:

[1] In Section 3, the frame I will be designated as the "Instrument" frame, and the other four frames as the faces A, B, C, and D of the RSDIMU instrument package.

$$
\begin{array}{|l}
T_{AI}, T_{BI}, T_{CI}, T_{DI}, T_{IA}, T_{IB}, T_{IC}, T_{ID} : \mathit{TransformMatrices} \\
\hline
T_{AI} = \begin{bmatrix} StoI(A,x)^T \\ StoI(A,y)^T \\ StoI(A,z)^T \end{bmatrix} \quad T_{BI} = \begin{bmatrix} StoI(B,x)^T \\ StoI(B,y)^T \\ StoI(B,z)^T \end{bmatrix} \\
T_{CI} = \begin{bmatrix} StoI(C,x)^T \\ StoI(C,y)^T \\ StoI(C,z)^T \end{bmatrix} \quad T_{DI} = \begin{bmatrix} StoI(D,x)^T \\ StoI(D,y)^T \\ StoI(D,y)^T \end{bmatrix} \\
T_{IA} = \begin{bmatrix} StoI(A,x) & StoI(A,y) & StoI(A,z) \end{bmatrix} \\
T_{IB} = \begin{bmatrix} StoI(B,x) & StoI(B,y) & StoI(B,z) \end{bmatrix} \\
T_{IC} = \begin{bmatrix} StoI(C,x) & StoI(C,y) & StoI(C,z) \end{bmatrix} \\
T_{ID} = \begin{bmatrix} StoI(D,x) & StoI(D,y) & StoI(D,z) \end{bmatrix}
\end{array}
$$

And, given, the following constant,

$$
pyrOffset == \frac{1}{\sqrt{6}} \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix}
$$

a series of transform functions can then be defined:

$$
\begin{array}{|l}
AtoI : \Re \rightarrow \mathit{Transformations} \\
\hline
\forall\, r : \Re \bullet (\\
\qquad AtoI(r).rotation = T_{AI} \wedge \\
\qquad AtoI(r).translation = r * pyrOffset)
\end{array}
$$

$$
\begin{array}{|l}
BtoI : \Re \rightarrow \mathit{Transformations} \\
\hline
\forall\, r : \Re \bullet (\\
\qquad BtoI(r).rotation = T_{BI} \wedge \\
\qquad BtoI(r).translation = r * pyrOffset)
\end{array}
$$

$$
\begin{array}{|l}
CtoI : \Re \rightarrow \mathit{Transformations} \\
\hline
\forall\, r : \Re \bullet (\\
\qquad CtoI(r).rotation = T_{CI} \wedge \\
\qquad CtoI(r).translation = r * pyrOffset)
\end{array}
$$

$$
\begin{array}{|l}
DtoI : \Re \rightarrow \mathit{Transformations} \\
\hline
\forall\, r : \Re \bullet (\\
\qquad DtoI(r).rotation = T_{DI} \wedge \\
\qquad DtoI(r).translation = r * pyrOffset)
\end{array}
$$

$$
\begin{array}{|l}
ItoA : \Re \rightarrow \mathit{Transformations} \\
\hline
\forall\, r : \Re \bullet (\\
\qquad ItoA(r).rotation = T_{IA} \wedge \\
\qquad ItoA(r).translation = -r * pyrOffset)
\end{array}
$$

$$
\begin{array}{|l}
ItoB : \Re \rightarrow Transformations \\
\hline
\forall\, r : \Re \bullet (\\
\qquad ItoB(r).rotation = T_{IB} \wedge \\
\qquad ItoB(r).translation = -r * pyrOffset)
\end{array}
$$

$$
\begin{array}{|l}
ItoC : \Re \rightarrow Transformations \\
\hline
\forall\, r : \Re \bullet (\\
\qquad ItoC(r).rotation = T_{IC} \wedge \\
\qquad ItoC(r).translation = -r * pyrOffset)
\end{array}
$$

$$
\begin{array}{|l}
ItoD : \Re \rightarrow Transformations \\
\hline
\forall\, r : \Re \bullet (\\
\qquad ItoD(r).rotation = T_{ID} \wedge \\
\qquad ItoD(r).translation = -r * pyrOffset)
\end{array}
$$

2.4 Pure Rotations

In many cases, we aren't really concerned with the translation portion of a transformation. It is therefore convenient to have a function for converting *TransformMatrices* directly into *Transformations*:

$$
\begin{array}{|l}
rotate : TransformMatrices \rightarrow Transformations \\
\hline
\forall\, M : TransformMatrices \bullet \\
\qquad rotate(M).rotation = M \wedge rotate(M).translation = zeros3D
\end{array}
$$

3 Physical Structure

The RSDIMU is composed of an instrument and a display. The display is discussed in Section 6. This section is devoted to the structure of the instrument package.

There are a number of frames associated with the RSDIMU problem. They are named as follows:

$$Frames == \{navigation, vehicle, instrument, A, B, C, D, \tilde{A}, \tilde{B}, \tilde{C}, \tilde{C}\}$$

The frames $A \mathrel{..} D$ are collectively known as the *sensor frames* of reference, and the frames $\tilde{A} \mathrel{..} \tilde{D}$ are the *measurement frames*. The measurement frames are paired with the sensor frames in the obvious manner:

$$measureFor == \{A \mapsto \tilde{A}, B \mapsto \tilde{B}, C \mapsto \tilde{C}, D \mapsto \tilde{D}\}$$

The acceleration due to the earth's gravity, measured in the *navigation* frame, is defined as:

$$
\begin{array}{|l}
g^N : Vectors3D \\
\hline
g^N = \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}
\end{array}
$$

3.1 Sensors

Each face of the RSDIMU instrument package contains a pair of linear sensors, known as the x and y sensor for that face (see Figure 2).

$$
\begin{array}{|l}
SensorNames : \mathbb{P}\, DirectionNames \\
\hline
SensorNames = \{x, y\}
\end{array}
$$

Each sensor produces a 12-bit digital output indicating a voltage proportional to the acceleration along the direction of the sensor. This 12 digit value is called a "count". The counts obtained from the sensors are proportional to the acceleration (in $\frac{meters}{sec^2}$) along the direction of that sensor.

$$
\begin{array}{l}
Counts == 0 \mathrel{..} 4096 \\
Accelerations == \Re
\end{array}
$$

Sensors are physical devices and produce noisy output. The largest standard deviation that would be expected from a functional sensor is given as the value *linstd*:

$$
\begin{array}{|l}
linstd : Counts
\end{array}
$$

Like most physical measurement devices, the sensors are sensitive to temperature changes:

$$Temperatures == \Re$$

$$
\begin{array}{|ll}
\multicolumn{2}{l}{Sensor} \\
linFail : boolean & \\
prevFailed : boolean & \\
offRaw : \mathrm{seq}\ Counts & \\
scale0, scale1, scale2 : \Re & \\
temp : Temperatures & \\
slope : \Re & \\
specificForce : Accelerations & \\
linOffset : Accelerations & \\
linNoise : boolean & \\
defective : boolean & \\
rawl : Counts & \\
lin : Accelerations & \\
\hline
slope = scale0 + scale1 * temp + scale2 * temp^2 & \\
specificForce = linOffset + slope * (rawl - 2048)/409.6 & (1) \\
lin = specificForce & \\
standardDev(offRaw) > 3 * linstd \Rightarrow defective & \\
prevFailed \Rightarrow linFail & \\
linNoise \Rightarrow linfail & (2) \\
linFail \Rightarrow defective & \\
\hline
\end{array}
$$

For each sensor, *linFail* represents the known state of the sensor (true if the sensor is known to be defective). Determining the proper value of this field is a major task of the calibration procedure

(Section 4). A sequence of values *offRaw* are used for this purpose, and also help to compute the proper *linOffset* used in converting the raw data into proper force measurements (the *specificForce*).

prevFailed is the state of the sensor prior to any calibration and estimation activities.

The temperature of the sensor is given by *temp*, and the scale factors $scale0$, $scale1$, and $scale2$ (determined at the factory) combine with temperature to determine the *slope*, which also is used to determine the *specificForce*.

If the noise (standard deviation) in the measurements *offRaw* exceeds $3 * linstd$, then the sensor will be marked as noisy (*linNoise* set to true) and therefore failed (*linFail* set to true) during the calibration procedure. Here, we model this by introducing a variable *defective* indicating the "true" state of the sensor. This variable will be hidden from the rest of the system.

Note that $linFail \Rightarrow defective$, but not $defective \Rightarrow linFail$, because a sensor may be defective without our knowing it until the calibration procedure has been completed. Note also that this simple check of the standard deviation is not the only way to determine that a sensor is defective. Sensors may also be marked as defective during the "edge-vector test" portion of the calibration, in which the consistency of sensors from adjacent faces is checked.

The current reading of the sensor is given as *rawl*. *lin* is the name given in the requirements for the current specific force.

Finally, we turn the above schema into a data type, hiding the "true" status:

$$Sensors == \{ \; Sensor \setminus defective \; \}$$

3.2 Faces

The instrument package is a semi-octahedron (a square-based pyramid) (Figure 1). It has four non-base faces, named by the sensor frame:

$$\begin{array}{|l} FaceNames : \mathbb{P}\, Frames \\ \hline FaceNames = \{A, B, C, D\} \end{array}$$

$$FaceStatus = \{nonOperational, partiallyOperational, completelyOperational\}$$

$$
\begin{array}{|l}
\hline
\textit{GeneralFace} \\
sensorFrame : FaceNames \\
measurementFrame : Frames \\
world : Worlds \\[4pt]
sensors : Sensors \rightarrow Sensors \\
misalign : MisalignmentAngles \\
temp : Temperatures \\[4pt]
normFace : Accelerations \\
status : FaceStatus \\[4pt]
specificForceMF : Points \\
specificForceSF : Points \\
measuredAccel : SensorNames \nrightarrow Accelerations \\
\hline
measurementFrame = measureFor(sensorFrame) \\[4pt]
\forall m : MisalignmentNames \bullet misalign(m) \approx \sin misalign(m) \\
sensors(x).temp = sensors(y).temp = temp \\[12pt]
world = \{ \\
\quad (sensorFrame, measurementFrame) \mapsto rotate(toMisaligned(misalign)), \\
\quad (measurementFrame, sensorFrame) \mapsto rotate(fromMisaligned(misalign))\} \\[4pt]
specificForceMF = \begin{bmatrix} sensors(x).specificForce \\ sensors(y).specificForce \\ normFace \end{bmatrix} \\
specificForceSF = transform(specificForceMF, measurementFrame, sensorFrame, world) \\
\hline
\end{array}
$$

Each face is named either A, B, C, or D, i.e., faces are named for one of the sensor frames. There is also a corresponding measurement frame representing the physical mounting of the sensors.

Each face contains two sensors, named the x and y sensors. There is an ideal position for each sensor, but the physical mounting may differ slightly. The differences are specified as the *misalign* angles for each sensor (see Section 2.3.2). The misalignment angles are assumed to be small enough (less than 5 degrees) for the sine of the angle to be approximately equal to the value of the angle expressed in radians.

The temperature of the face, *temp*, determines the temperatures of the sensors mounted on the face.

During the calibration procedure, the face is subjected to a known acceleration *normFace* normal (perpendicular) to the face.

The total specific force vector acting on this face, as measured by this face's sensors, is given in the *measurement* and *sensor* frames by *specificForceMF* and *specificForceSF*, respectively. The *measuredAccel* is the quantity used in the actual acceleration estimation procedure.

$$
\begin{array}{|l}
\hline
\textit{CompletelyOperationalFace} \\
GeneralFace \\
\hline
status = completelyOperational \\
\neg\ sensor(x).linfail \\
\neg\ sensor(y).linfail \\[4pt]
measuredAccel = \lambda\ d : SensorNames \bullet specificForceSF.coord(d) \\
\hline
\end{array}
$$

The operational state of the face is given by *status*. A face is completely operational only if both of its sensors are (believed to be) functional. For a completely operational face, the estimation process uses the compensated-for-misalignment measurements from the *sensor* frame.

$$
\begin{array}{|l}
\hline
\textit{PartiallyOperationalFace} \\
\textit{GeneralFace} \\
\hline
status = partiallyOperational \\
sensor(x).linfail \Leftrightarrow \neg\, sensor(y).linfail \\
measuredAccel = \lambda\, d : SensorNames \mid \neg\, sensor(d).linFail \bullet specificForceMF.coord(d) \\
\hline
\end{array}
$$

A face is partially operational if exactly one of its sensors is failed. For a partially operational face, the estimation process uses the uncompensated measurement, in the *measurement* frame, from the one operating *sensor*.

$$
\begin{array}{|l}
\hline
\textit{NonOperationalFace} \\
\textit{GeneralFace} \\
\hline
status = nonOperational \\
sensor(x).linfail \wedge sensor(y).linfail \\
measuredAccel = \varnothing \\
\hline
\end{array}
$$

A face is non-operational if both sensors are (known to be) failed.

$$
\begin{array}{l}
Face \mathrel{\widehat=} GeneralFace \wedge \\
\quad (CompletelyOperationalFace \vee PartiallyOperationalFace \vee NonOperationalFace)
\end{array}
$$

$$Faces == \{\ Face\ \}$$

3.3 The Instrument

The four faces together constitute the instrument portion of the RSDIMU. The major constraint at this level is simply the assignment of appropriate names to each face and the description of the arrangement of the four faces into the semi-octahedron.

$$
\begin{array}{|l}
\hline
\textit{Instrument} \\
face : FaceNames \rightarrow Faces \\
obase : \Re \\
iworld : Worlds \\
nonOpFaces : \mathbb{F}\, FaceNames \\
\hline
\forall f : FaceNames \bullet face(f).measurementFrame = f \\
nonOpFaces = \{f : FaceNames \mid face(f).status = nonOperational\} \\
iworld = \bigcup\{face(A).world, face(B).world, face(C).world, face(D).world\} \\
\quad \cup\{ \quad (instrument, A) \mapsto ItoA(obase), \\
\qquad (instrument, B) \mapsto ItoB(obase), \\
\qquad (instrument, C) \mapsto ItoC(obase), \\
\qquad (instrument, D) \mapsto ItoD(obase), \\
\qquad (A, instrument) \mapsto AtoI(obase), \\
\qquad (B, instrument) \mapsto BtoI(obase), \\
\qquad (C, instrument) \mapsto CtoI(obase), \\
\qquad (D, instrument) \mapsto DtoI(obase)\} \\
\hline
\end{array}
$$

$obase$ is the length of the base of the pyramid.

$nonOpFaces$ is the set of faces that have been determined to be non-operational (i.e., both sensors on that face have failed).

Closely related to the *Instrument* schema is the *Vehicle* schema:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{Vehicle}} \\
\hline
Instrument \\
\psi_I, \theta_I, \phi_I : Angles \\
vworld : Worlds \\
\hline
vworld = iworld \cup \\
\quad \{(vehicle, instrument) \mapsto rotate(yaw\_pitch\_roll(\psi_I, \theta_I, \phi_I)), \\
\quad (instrument, vehicle) \mapsto rotate(roll\_pitch\_yaw(\psi_I, \theta_I, \phi_I))\} \\
\hline
\end{array}
$$

This adds to the instrument state the description of the rotation of the instrument package with respect to the aircraft it flies in.

3.4 The RSDIMU State

Finally, we can describe the overall state of the RSDIMU system.

$$SystemStatus == \{normal, analytic, undefined\}$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{RSDIMUGeneral}} \\
\hline
Vehicle \\
FourChannels \\
EdgeSet \\
Display \\
\psi_v, \theta_v, \phi_v : Angles \\
world : Worlds \\
acceleration : Vectors3D \\
status : SystemStatus \\
sysStatus : boolean \\
\hline
world = vworld \cup \\
\quad \{(navigation, vehicle) \mapsto rotate(yaw\_pitch\_roll(\psi_v, \theta_v, \phi_v)), \\
\quad (vehicle, navigation) \mapsto rotate(roll\_pitch\_yaw(\psi_v, \theta_v, \phi_v))\} \\
Display.faces = Vehicle.Instrument.face \\
Display.bestEst = acceleration \\
\hline
\end{array}
$$

The three angles give the orientation of the vehicle with respect to the ground (*navigation* frame). The *acceleration* is the current least squares estimate of the vehicle acceleration. Computing this is, in a sense, the primary goal of the RSDIMU package.

The *channel*s provide alternate estimates of the vehicle acceleration. Their definition is deferred until Section 5.2. *EdgeSet* contains information related to the inter-Face (not "interface") edges and will be defined in Section 4.2.2. Similarly, all discussion of the *Display* is deferred to Section 6.

The general RSDIMU state is divided into three major cases by the *status* variable. Under *normal* circumstances, there are more than enough sensors to permit the computation of the *acceleration*. If enough sensors fail, we may be able to compute the *acceleration* via *analytic* means, with no redundancy. If any more sensors fail, then we cannot compute the *acceleration* and the system status is *undefined*.

The *sysStatus* field indicates whether at least two faces in the RSDIMU instruement are completely operational and their edge of intersection satisfies the "edge test" described in Section 4.2.2.

$$
\begin{array}{|l}
\hline
\textit{RSDIMU\_Normal} \\
\textit{RSDIMUGeneral} \\
\hline
\textit{status} = \textit{normal} \\
\#\{ f : \textit{FaceNames};\ d : \textit{SensorNames} \mid \neg\, \textit{face}(f).\textit{sensor}(d).\textit{linFail} \} > 3 \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{RSDIMU\_Analytic} \\
\textit{RSDIMUGeneral} \\
\hline
\textit{status} = \textit{analytic} \\
\#\{ f : \textit{FaceNames};\ d : \textit{SensorNames} \mid \neg\, \textit{face}(f).\textit{sensor}(d).\textit{linFail} \} = 3 \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{RSDIMU\_Undefined} \\
\textit{RSDIMUGeneral} \\
\hline
\textit{status} = \textit{undefined} \\
\#\{ f : \textit{FaceNames};\ d : \textit{SensorNames} \mid \neg\, \textit{face}(f).\textit{sensor}(d).\textit{linFail} \} < 3 \\
\hline
\end{array}
$$

$$
\textit{RSDIMU} \mathrel{\widehat{=}} \textit{RSDIMU\_Normal} \lor \textit{RSDIMU\_Analytic} \lor \textit{RSDIMU\_Undefined} \tag{3}
$$

There are two major state-transition schemas for the RSDIMU. These are *Calibration* and *EstimateAcceleration*, defined in Sections 4 and 5, respectively. Many of the quantities specified within the RSDIMU and related schemas are unchanged by these major state transitions. It is convenient, therefore, to introduce the following schema that will save us the trouble of listing these invariant quantities within each of the state transitions still to come.

$$
\begin{array}{|l}
\hline
\textit{FixedQuantities} \\
\Delta RSDIMU \\
\hline
world' = world \\
\psi'_I = \psi_I \\
\theta'_I = \theta_I \\
\phi'_I = \phi_I \\
obase' = obase \\
\\
\forall f : FaceNames \bullet \\
\quad (face(f).sensorFrame' = face(f).sensorFrame \land \\
\quad face(f).misalign' = face(f).misalign \land \\
\quad face(f).temp' = face(f).temp \land \\
\quad face(f).normFace' = face(f).normFace) \\
\\
\forall f : FaceNames;\ d : SensorNames \bullet \\
\quad (face(f).offRaw' = face(f).offRaw \land \\
\quad face(f).sensor(d).scale0' = face(f).sensor(d).scale0 \land \\
\quad face(f).sensor(d).scale1' = face(f).sensor(d).scale1 \land \\
\quad face(f).sensor(d).scale2' = face(f).sensor(d).scale2 \land \\
\quad face(f).sensor(d).rawl' = face(f).sensor(d).rawl \land \\
\quad face(f).sensor(d).prevFailed' = face(f).sensor(d).prevFailed) \\
\\
DMode' = DMode \\
\hline
\end{array}
$$

4 Calibration and Failure Detection

This section describes the process of calibrating sensors and checking them for possible failure. These activities divide naturally into two groups, the "at-rest" and "in-flight" procedures.

4.1 Vehicle At Rest

When the vehicle is at rest, the only force acting upon the sensors is the gravity. Because the force due to gravity is a known quantity, it can be employed to help calibrate the sensors. Also, the controlled, static environment in effect when the vehicle is at rest permits us to take multiple sensor readings and analyze them for possibly excessive noise.

4.1.1 Calibration - Calculating Sensor Offsets

In this procedure, the projection of the gravitational acceleration vector along each sensor is computed. Then, the standard formula (equation 1, page 11) relating sensor readings to specific force allows us to solve for the *linOffset* for that sensor.

When the vehicle is at rest, the only force acting upon it is the force of gravity, $-g^N$. We can transform this vector into any frame defined within the RSDIMU world:

$$
\begin{array}{|l}
\hline
\textit{AtRestSF} \\
\Xi RSDIMU \\
Atrestp : FaceNames \rightarrow Points \\
\hline
Atrestp = \lambda f : FaceNames \bullet transform(-g^N, N, f, world) \\
\hline
\end{array}
$$

$$atRestAcc == \lambda f : FaceNames;\ d : DirectionNames \mid AtRestSF \bullet Atrestp(f).coord(d)$$

Next, we introduce a pair of "utility" functions for computing averages and standard deviations:

$$avg == \lambda\, c : \mathrm{seq}\ Counts \bullet \frac{1}{\# c}\sum_{i=1}^{\# c} c(i)$$

$$standardDev == \lambda\, c : \mathrm{seq}\ Counts \bullet \sqrt{\frac{1}{\# c}\sum_{i=1}^{\# c}(c(i) - avg(c))^2}$$

Now the process of determining offsets for each sensor is given by:

$$\begin{array}{|l}
\textit{DetermineOffsets} \\
\hline
\Delta RSDIMU \\
FixedQuantities \\
\hline
world(N, V).translation = Zero3D \\
acceleration = Zero3D \\
\\
\forall f : FaceNames;\ d : SensorNames;\ m : Frames \mid m = measureFor(f) \bullet \\
\quad face(f).sensor(d).linOffset' = \\
\qquad atRestAcc(m, d) - slope * \frac{avg(face(f).sensor(d).offRaw) - 2048}{409.6} \\
acceleration = acceleration' \\
sysStatus = sysStatus' \\
\forall f : FaceNames;\ d : SensorNames \bullet \\
\quad (face(f).linFail' = face(f).linFail \land \\
\quad face(f).sensor(d).linNoise' = face(f).sensor(d).linNoise) \\
\hline
\end{array}$$

A precondition of this schema is that the vehicle must be at rest at the origin of the navigation frame. Under these circumstances, we can compute the *linOffset* value for each sensor by substituting the average of the calibration input readings (*offRaw*) for the input *rawl* of equation (1).

The remaining conditions merely indicate that this process leaves all but the *linOffset* values unchanged.

4.1.2 Failures From Noisy Sensors

There are three distinct sources of sensor failure information in the RSDIMU. First, we have the record of failures of the given instrument on prior tests. This is denoted by the value of *linFail* for each sensor prior to the current calibration and failure detection process.

Second, some sensors can be determined to have failed by examining the amount of noise in their readings during the offset determination process. If this exceeds a certain threshold, then the sensor is marked as noisy and failed. This noise detection is the subject of this Section.

The third and most complex source of sensor failure information is the edge-vector test, which will be covered in Section 4.2

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{CheckForNoise}} \\
\Delta RSDIMU \\
FixedQuantities \\
\hline
world(N, V).translation = Zero3D \\
acceleration = Zero3D \\
\\
\forall f : FaceNames;\ d : SensorNames;\ m : Frames \mid m = measureFor(f) \\
\quad \bullet\ face(f).sensor(d).linNoise' \Leftrightarrow standardDev(face(f).sensor(d).offRaw) > 3 * linstd \\
acceleration' = acceleration \\
sysStatus' = sysStatus \\
\forall f : FaceNames;\ d : SensorNames\ \bullet \\
\quad face(f).linOffset' = face(f).linOffset \\
\hline
\end{array}
$$

Similar to the *DetermineOffsets* schema, this schema shows the process of checking noise in sensor measurements taken with the vehicle at rest. If the standard deviation of these measurements exceeds $3 * linstd$, then the sensor is marked as noisy (and, by implication of equation 2 on page 11, as failed).

4.2 In-Flight Failure Detection

In-flight readings of the sensor array are checked for consistency by projecting the readings of sensors from adjacent faces onto the direction representing the edge of intersection between the two faces. In an ideal system, the two resulting force values would be identical. In practice, some deviation is expected due to physical measurement error. If that difference exceeds a predetermined threshold, however, it indicates a malfunction by at least one sensor on the affected faces.

In the edge-vector test, we must first determine what faces (if any) have failed. Then, from among the malfunctioning faces, we determine which sensor is not functioning properly.

To determine which faces are bad, we consider every pair of faces. Note that there is exactly one edge in the semioctahedron coincident to any given pair of faces. For the purpose of illustration, let us consider the faces A and B and let us call the edge of intersection between these faces AB. Now we take the accelerometer measurements in A and project them onto AB. We also take the accelerometer measurements in B and project them onto AB. The next step is to take the difference between these two projections. If both faces are functioning properly, the difference should be below some threshold value. We repeat this process for every pair of faces in the semioctahedron. A face is considered to have failed if all comparisons involving that face are out of tolerance.

Once we have found the bad face,[2] we need to determine which sensor in that face has failed. We do this by computing the least squares estimate of the specific force in the Instrument Frame. We use only the faces that have not been labeled failed to compute this estimate. Once we have computed this estimate of specific force, we project it onto the axis of each sensor in the questionable face. We now compare this estimated specific force on the sensor axis to the actual sensor measurement. If the difference between these two is greater than a predetermined threshold value,

After obtaining the (possibly) new status of the sensors, we update the information in the linFail attribute in Sensor to reflect any new failures. We also report our results in the status attribute of the Face schema.

[2] The possibility of a simultaneous in-flight failure of more than one previously operational sensor is explicitly discounted in [1], presumably because the probability of such an event is negligible.

4.2.1 The Test Threshold

The integer *nsigt* controls the sensitivity of the edge test.

$$nsigt : 2 \mathinner{\ldotp\ldotp} 7$$

EdgeThreshold

$RSDIMU$
$goodSlopes : \mathbb{P}\,\Re$
$\sigma_s : \Re$
$\delta : \Re$

$goodSlopes = \{\, f : FaceNames;\; d : SensorNames \mid \neg\, face(f).sensor(d).linFail$
$\quad \bullet\, face(f).sensor(d).slope \,\}$

$$\sigma_s = \frac{linstd}{409.6} * \frac{1}{\#goodSlopes} \sum_{s \in goodSlopes} s$$

$$\delta = \sqrt{2} * nsigt * \sigma_s$$

$$\delta == EdgeThreshold.\delta$$

The threshold δ for the edge test is defined to be *nsigt* times the maximum acceptable noise level, *linstd*, expressed in volts, times the average of the acceleration/voltage slopes among those sensors believed to be operational.

4.2.2 Edges

The intersection of each pair of instrument faces defines a unique edge, which we name by giving the face pairs:

$$EdgeNames == \{\langle A, B\rangle, \langle A, C\rangle, \langle A, D\rangle, \langle B, C\rangle, \langle B, D\rangle, \langle C, D\rangle\}$$

We will need to map vectors in a face's sensor frame onto an edge of that face. To do so, we divide the possible combinations of faces and edges into four cases, the edge between a face and the adjacent face reached moving clockwise around the instrument, the edge between a face and the adjacent face reached moving counter-clockwise around the instrument, the edge between a face and a higher-lettered opposite face, and the edge between a face and a lower-lettered opposite face. These cases are detailed in the following set definitions.

$$ClockwiseEdgeFace == \{(A, \langle A, B\rangle), (B, \langle B, C\rangle), (C, \langle C, D\rangle), (D, \langle D, A\rangle)\}$$
$$CounterclockwiseEdgeFace == \{(B, \langle A, B\rangle), (C, \langle B, C\rangle), (D, \langle C, D\rangle), (A, \langle D, A\rangle)\}$$
$$OppositeEdgeFace1 == \{(A, \langle A, C\rangle), (B, \langle B, D\rangle)\}$$
$$OppositeEdgeFace2 == \{(C, \langle A, C\rangle), (D, \langle B, D\rangle)\}$$

Each face is an equilateral triangle. The sensor frame of reference has the sensor x and y axes mounted as shown in Figure 2. The projection function can therefore be derived as:

$$
\begin{array}{l}
mapToEdge : Points \times FaceNames \times EdgeNames \rightarrow\!\!\!\!\!+ \Re \\
\hline
mapToEdge = \{\ p : Points;\ f : FaceNames;\ e : EdgeNames \\
\qquad \mid (f, e) \in ClockwiseEdgeFace \bullet (p, f, e) \mapsto p.x \cos(15) + p.y \cos(75)\ \} \\
\quad \cup \qquad \{\ p : Points;\ f : FaceNames;\ e : EdgeNames \\
\qquad \mid (f, e) \in CounterclockwiseEdgeFace \bullet (p, f, e) \mapsto p.x \cos(75) + p.y \cos(15)\ \} \\
\quad \cup \qquad \{\ p : Points;\ f : FaceNames;\ e : EdgeNames \\
\qquad \mid (f, e) \in OppositeEdgeFace1 \bullet (p, f, e) \mapsto p.x \cos(45) + p.y \cos(-45)\ \} \\
\quad \cup \qquad \{\ p : Points;\ f : FaceNames;\ e : EdgeNames \\
\qquad \mid (f, e) \in OppositeEdgeFace2 \bullet (p, f, e) \mapsto p.x \cos(-45) + p.y \cos(45)\ \}
\end{array}
$$

For each edge, we will require the following information.

$$
\begin{array}{l}
\textit{Edge} \\
\hline
RSDIMUGeneral \\
name : EdgeNames \\
diff : \Re \\
bad : boolean \\
\hline
diff = mapToEdge(name(1), name, face(name(1)).specificForceSF) - \\
\qquad mapToEdge(name(2), name, face(name(2)).specificForceSF) \\
bad = \mid diff \mid > \delta \\
\hline
\end{array}
$$

$$Edges == \{\ Edge\ \}$$

$name$ indicates the name of the edge, and therefore the names of the faces that define the edge.

$diff$ is the difference in the specific force estimates, projected along the edge, by each of the two incident faces.

bad is true if and only if $diff$ exceeds the allowable threshold.

The RSDIMU contains these edges.

$$
\begin{array}{l}
\textit{EdgeSet} \\
\hline
edge : EdgeNames \rightarrow Edges \\
\hline
\end{array}
$$

4.2.3 Checking The Edge-Vectors

Any face that was previously believed to be completely operational should now be marked as failed (partially operational) if it fails the edge vector test on all edges shared with other completely operational faces.

$$
\begin{array}{l}
\textit{FaceFailsEdgeTest} \\
\hline
\Xi RSDIMU \\
f? : FaceNames \\
fails! : boolean \\
\hline
fails! = (face(f?).status = completelyOperational) \wedge \\
\quad \forall f2 : FaceNames \mid face(f2).status = completelyOperational \bullet \\
\qquad (\langle f?, f2 \rangle \in EdgeNames \Rightarrow edge(\langle f?, f2 \rangle).bad) \wedge \\
\qquad (\langle f2, f? \rangle \in EdgeNames \Rightarrow edge(\langle f2, f? \rangle).bad) \\
\hline
\end{array}
$$

$$faceFailsEdgeTest == \lambda f : FaceNames \mid FaceFailsEdgeTest \wedge f = f? \bullet fails!$$

4.2.4 Isolating Sensor Failure

The final step in the calibration and failure detection process is to determine which, if any, individual sensors have failed. We need no re-examine sensors that have been previously determined to have failed (either prior to the calibration process or because we have determined them to have been excessively noisy. Similarly, we do not further examine sensors on completely operational faces that have passed this latest edge test. This leaves two cases to be checked:

- Check both sensors in any face that was previously believed to be completely operational (both sensors OK) but that has just failed the edge test, and
- Check the previously believed-to-be-good sensor in any partially operational faces (only one sensor working).

In each case, the sensors to be checked are tested by comparing their output to the projection along the sensor direction of the least squares estimate of the vehicle acceleration. The computation of this estimate is discussed in Section 5.

The following function determines if a sensor is compatible with the least squares estimate of the vehicle acceleration.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\_\; \mathit{CheckSensorAgainstLSQ}\; \_\_\_\_\_\_\_\_} \\
\Xi \mathit{RSDIMU} \\
f? : \mathit{FaceNames} \\
d? : \mathit{SensorNames} \\
\mathit{fails}! : \mathit{boolean} \\
\mathit{proj} : \Re \\
\hline
\mathit{proj} = \mathit{transform}(\mathit{acceleration}, \mathit{navigation}, \mathit{measureFor}(f?), \mathit{world}).\mathit{coord}(d?) \\
\mathit{fails}! = \mid \mathit{proj} - \mathit{face}(f?).\mathit{sensor}(d?).\mathit{specificForce} \mid > \delta \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\mathit{sensorDeviatesFromLSQ} == \lambda f : \mathit{FaceNames};\ d : \mathit{SensorNames} \\
\quad \mid \mathit{CheckSensorAgainstLSQ} \wedge f = f? \wedge d = d? \\
\quad \bullet\ \mathit{fails}!
\end{array}
$$

proj is the projection of the estimated vehicle acceleration into the measurement frame appropriate to the sensor. $\mathit{fails}!$ is true when the difference between proj and the actual sensor reading exceeds the threshold δ.

Now the process of isolating sensor failures can be separated into the following cases:

$$
\begin{array}{l}
\textbf{CheckFailingFace} \\
\hline
\Delta RSDIMU \\
FixedQuantities \\
f? : FaceNames \\
\hline
face(f?).status = completelyOperational \\
faceFailsEdgeTest(f?) \\[4pt]
face(f?).sensor(x).linFail' = sensorDeviatesFromLSQ(f?, x) \\
face(f?).sensor(y).linFail' = sensorDeviatesFromLSQ(f?, y) \\[4pt]
acceleration' = acceleration \\
status' = status \\[4pt]
\forall f : FaceNames;\ d : SensorNames \bullet \\
\quad (face(f).linOffset' = face(f).linOffset \wedge \\
\quad face(f).sensor(d).linNoise' = face(f).sensor(d).linNoise) \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textbf{CheckPartialFaceX} \\
\hline
\Delta RSDIMU \\
FixedQuantities \\
f? : FaceNames \\
\hline
face(f?).status = partiallyOperational \\
\neg\, face(f?).sensor(x).linFail \\
face(f?).sensor(x).linFail' = sensorDeviatesFromLSQ(f?, x) \\
face(f?).sensor(y).linFail' = face(f?).sensor(y).linFail \\[4pt]
acceleration' = acceleration \\
status' = status \\[4pt]
\forall f : FaceNames;\ d : SensorNames \bullet \\
\quad (face(f).linOffset' = face(f).linOffset \wedge \\
\quad face(f).sensor(d).linNoise' = face(f).sensor(d).linNoise) \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textbf{CheckPartialFaceY} \\
\hline
\Delta RSDIMU \\
FixedQuantities \\
f? : FaceNames \\
\hline
face(f?).status = partiallyOperational \\
\neg\, face(f?).sensor(y).linFail \\
face(f?).sensor(y).linFail' = sensorDeviatesFromLSQ(f?, y) \\
face(f?).sensor(x).linFail' = face(f?).sensor(x).linFail \\[4pt]
acceleration' = acceleration \\
status' = status \\[4pt]
\forall f : FaceNames;\ d : SensorNames \bullet \\
\quad (face(f).linOffset' = face(f).linOffset \wedge \\
\quad face(f).sensor(d).linNoise' = face(f).sensor(d).linNoise) \\
\hline
\end{array}
$$

Then, combined with the following schemas:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{FaceCompletelyOperational}} \\
\Xi RSDIMU \\
f? : FaceNames \\
\hline
face(f?).status = completelyOperational \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{FacePartiallyOperational}} \\
\Xi RSDIMU \\
f? : FaceNames \\
\hline
face(f?).status = partiallyOperational \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{FaceNonOperational}} \\
\Xi RSDIMU \\
f? : FaceNames \\
\hline
face(f?).status = nonOperational \\
\hline
\end{array}
$$

$$
\begin{aligned}
& CheckSensorsInFace \mathrel{\widehat=} (FaceCompletelyOperational \land FaceFailsEdgeTest \land CheckFailingFace) \\
& \quad \lor (FacePartiallyOperational \land (CheckPartialFaceX \lor CheckPartialFaceY)) \\
& \quad \lor FaceCompletelyOperational \land \neg FaceFailsEdgeTest) \\
& \quad \lor FaceNonOperational
\end{aligned}
$$

$$checkSensorsInFace == \lambda f : FaceNames \mid CheckSensorsInFace \land f = f? \bullet true$$

Finally, we must apply this process to each face in the instrument.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{IsolateSensorFailures}} \\
\Delta RSDIMU \\
FixedQuantities \\
\Xi Display \\
\hline
\forall f : FaceNames \bullet checkSensorsInFace(f) \\
sysStatus' = \exists f1, f2 : FaceNames \mid \langle f1, f2 \rangle \in EdgeNames \bullet \\
\qquad face(f1).status = face(f2).status = completelyOperational \land \\
\qquad \neg\ edge(\langle f1, f2 \rangle).bad) \\
\\
acceleration' = acceleration \\
status' = status \\
\\
\forall f : FaceNames;\ d : SensorNames \bullet \\
\qquad (face(f).linOffset' = face(f).linOffset \land \\
\qquad face(f).sensor(d).linNoise' = face(f).sensor(d).linNoise) \\
\hline
\end{array}
$$

4.2.5 Calibration and Failure Detection

The entire calibration and failure detection procedure consists of

$$Calibration \mathrel{\widehat=} DetermineOffsets \mathbin{⨾} CheckForNoise \mathbin{⨾} BestEstimateAccleration \mathbin{⨾} IsolateSensorFailures$$

5 Vehicle Acceleration Estimation

5.1 Overview

Given the set of operational sensors determined by the sensor failure detection and isolation test, either an overdefined, exactly defined, or underdefined system of equations will exist. In the first case, a least squares estimate is calculated and in the second case, an analytic solution is calculated.

Actually, five estimates of the vehicle acceleration are produced. One of these is the *bestEst*, an estimate produced through the use of all operational sensors. The other four estimates are produced using different pairs of completely operational faces (if possible).

5.2 Channels

The four face-pair estimates are said to be produced on one of four *Channels*. In an ideal instrument, all four faces would be available for this purpose, leading to 360 possible assignments of non-duplicate pairs of faces to the four channels.

There are four channels, numbered 1 through 4:

$$ChannelNums == \{1, 2, 3, 4\}$$

The possible sets of two faces are numbered from 1 to 6. The empty set is numbered 0. Each channel is assigned a set of faces, and the corresponding pair number is part of the channel state.

$$PairNums : \mathbb{F}\,\mathbb{N}$$

$$PairNums = \{0, 1, 2, 3, 4, 5, 6\}$$

The following relation shows the mapping of pair numbers to pairs of faces.

$$facePairs : PairNums \rightarrow \mathbb{F}\,FaceNames$$

$$\begin{aligned} facePairs = \{ & 0 \mapsto \{\} \\ & 1 \mapsto \{A, B\} \\ & 2 \mapsto \{A, C\} \\ & 3 \mapsto \{A, D\} \\ & 4 \mapsto \{B, C\} \\ & 5 \mapsto \{B, D\} \\ & 6 \mapsto \{C, D\}\} \end{aligned}$$

$$channelAssignments : \mathbb{F}\,FaceNames \rightarrow \operatorname{seq} PairNums$$

$$\begin{aligned} channelAssignments = \{ & \\ & \{\} \mapsto \langle 1, 4, 6, 3 \rangle \\ & \{A\} \mapsto \langle 0, 4, 6, 5 \rangle \\ & \{B\} \mapsto \langle 2, 0, 6, 3 \rangle \\ & \{C\} \mapsto \langle 1, 5, 0, 3 \rangle \\ & \{D\} \mapsto \langle 1, 4, 2, 0 \rangle \\ & \{A, B\} \mapsto \langle 0, 0, 6, 0 \rangle \\ & \{A, C\} \mapsto \langle 0, 5, 0, 0 \rangle \\ & \{A, D\} \mapsto \langle 0, 4, 0, 0 \rangle \\ & \{B, C\} \mapsto \langle 0, 0, 0, 3 \rangle \\ & \{B, D\} \mapsto \langle 2, 0, 0, 0 \rangle \\ & \{C, D\} \mapsto \langle 1, 0, 0, 0 \rangle\} \end{aligned}$$

Channels are assigned a set of faces depending upon which faces are nonoperational. The pair of faces assigned to each of the four channels can be represented as a sequence of pair numbers. The number of the pair assigned to channel 1 is the first value of the sequence, that of channel 2, the second, and so on. *channelAssignments* shows the mapping from sets of nonoperational faces to sequences of pair numbers. For example, if face C is nonoperational then Channel 1 has *PairNums* 1, Channel 2 has pair 5, Channel 3 has pair 0 (no faces assigned), and Channel 4 has pair 3. Note that if more than two faces fail, the state of the system is undefined.

The schema for a channel is simply:

Channel

$status : SystemStatus$
$acceleration : Vectors3D$
$chanface : PairNums$

and a channel type is:

$$Channels == \{Channel\}$$

The *RSDIMU_General* schema includes a schema *FourChannels*, which we can now define:

FourChannels

$channel : \mathrm{seq}\ Channels$

$\mathrm{dom}\ channel = ChannelNums$

5.3 Basic Calculations

Compensated accelerometer measurements are transformed from the Sensor Frame to the Measurement Frame. Normally, compensated accelerometer measurements are used to make an acceleration estimate. However, if one sensor on a face has failed, then an uncompensated (Measurement Frame) measurement is used for the remaining sensor.

The operating sensors being used in any vehicle acceleration estimation (best or channel) each contribute an equation to the system

$$\bar{y} = C\bar{f}^I \tag{4}$$

where $\bar{y}$ is a vector of *measuredAccel* values for each contributing sensor, $\bar{f}^I$ is the unknown force (acceleration) on the instrument, and C is a matrix representing the transformation from the *sensor* frame for each contributing sensor's direction into the *instrument* frame.

$$EquationIDs == FaceNames \times DirectionNames$$

Each equation in the system (4) is associated with a particular sensor, which in turn is identified by a face-direction pair.

DoCreateYmap

$\Xi RSDIMU$
$fnames? : \mathbb{F}\ FaceNames$
$ymap! : EquationIDs \rightarrowtail\!\!\!\!\!\!+ Accelerations$

$ymap! = \lambda f : FaceNames;\ d : DirectionNames$
$\quad \mid f \in fnames? \wedge d \in \mathrm{dom}\, face(f).measuredAccel$
$\quad \bullet\ face(f).measuredAccel(d)$

The schema is put into a functional form:

$$createYmap = \lambda fnames : \mathbb{F}\, FaceNames \mid DoCreateYmap \land fnames = fnames? \bullet ymap!$$

Given a set of faces, *createYmap* returns a relation that maps face names and sensors to accelerometer measurements. Nonoperational sensors do not contribute a measurement value. In the case of a partially operational face, the value used for the operational sensor is not compensated for misalignment.

$$
\begin{array}{l}
\textit{DoCreateCmap} \\
\hline
\Xi RSDIMU \\
fnames? : \mathbb{F}\, FaceNames \\
map : EquationIDs \pfun Vectors3D \\
cmap! : DirectionNames \rightarrow (EquationIDs \pfun \Re) \\
\hline
map = \lambda f : FaceNames;\ d : SensorNames \\
\quad \mid f \in fnames? \land \neg\, face(f).sensor(d).linFail \\
\quad \bullet\ StoI(f, d) \\
cmap! = map^{T} \\
\hline
\end{array}
$$

The schema is put into a functional form:

$$createCmap = \lambda fnames : \mathbb{F}\, FaceNames \mid DoCreateCmap \land fnames = fnames? \bullet cmap!$$

Given a set of faces, *createCmap* returns a relation mapping face names and directions to transformation vectors. As before, nonoperational sensors are not transformed and do not contribute to the final composition of the output set. The entries are obtained from the *StoI* transformation defined in Section 2.3.3.

Using the building blocks above, we can define functions to be used in calculating the vehicle acceleration.

$$
\begin{array}{l}
\textit{AnalyticSolution} \\
\hline
\Delta RSDIMU \\
\Xi Vehicle \\
\Xi FourChannels \\
usingFaces? : \mathbb{P}\, FaceNames \\
acceleration! : Vectors3D \\
status! : SystemStatus \\
C : DirectionNames \rightarrow (EquationIDs \pfun \Re) \\
y : EquationIDs \pfun Accelerations \\
\hline
C = createCmap(usingFaces?) \\
y = createYmap(usingFaces?) \\
\#\operatorname{dom} y = 3 \\
\\
acceleration! = C^{-1} y \\
status! = analytic \\
\\
world' = world \\
\psi'_v = \psi_v \\
\theta'_v = \theta_v \\
\phi'_v = \phi_v \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{LSQSolution} \\
\Delta RSDIMU \\
\Xi Vehicle \\
\Xi FourChannels \\
usingFaces? : \mathbb{P}\, FaceNames \\
acceleration! : Vectors3D \\
status! : SystemStatus \\
C : DirectionNames \rightarrow (EquationIDs \nrightarrow \Re) \\
y : EquationIDs \nrightarrow Accelerations \\
\hline
C = createCmap(usingFaces?) \\
y = createYmap(usingFaces?) \\
\#\,\mathrm{dom}\, y > 3 \\
acceleration! = (C^T C)^{-1} Cy \\
status! = normal \\
world' = world \\
\psi'_v = \psi_v \\
\theta'_v = \theta_v \\
\phi'_v = \phi_v \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{UndefinedSolution} \\
\Delta RSDIMU \\
\Xi Vehicle \\
\Xi FourChannels \\
usingFaces? : \mathbb{P}\, FaceNames \\
acceleration! : Vectors3D \\
status! : SystemStatus \\
y : EquationIDs \nrightarrow Accelerations \\
\hline
y = createYmap(usingFaces?) \\
\#\,\mathrm{dom}\, y < 3 \\
acceleration! = zero3D \\
status! = undefined \\
\forall f : FaceNames;\ d : SensorNames \bullet \\
\quad (face(f).linOffset' = face(f).linOffset\ \wedge \\
\quad face(f).linFail' = face(f).linFail\ \wedge \\
\quad face(f).sensor(d).linNoise' = face(f).sensor(d).linNoise) \\
world' = world \\
\psi'_v = \psi_v \\
\theta'_v = \theta_v \\
\phi'_v = \phi_v \\
\hline
\end{array}
$$

These schemas and others to follow assume prior definitions of matrix transpose, matrix inverse and matrix multiplication operations.

AnalyticSolution and *LSQSolution* return the vehicle acceleration represented in the Instrument Frame. *GetSolution* uses whichever of these is applicable, depending upon the number of of equations in the system 4 (which, in turn, reflects the number of operational sensors within the chosen faces).

$GetSolution \mathrel{\widehat=} LSQSolution \lor AnalyticSolution \lor UndefinedSolution$
$getSolution = \lambda\, usingFaces : \mathbb{P}\, FaceNames \mid GetSolution \land usingFaces? = usingFaces$
$\quad \bullet\ (acceleration!, status!)$

5.4 Acceleration Estimation

BestEstimateAcceleration

$\Delta RSDIMU$
$\Xi Display$
$\Xi Vehicle$
acc^I

$(acc^I, status') = getSolution(\{A, B, C, D\})$
$acceleration' = transform(acc^I, instrument, navigation, world) + g^N$

$world' = world$
$\psi'_v = \psi_v$
$\theta'_v = \theta_v$
$\phi'_v = \phi_v$

The solution for the acceleration, computed in the *instrument* frame, must be transformed into the Navigation Frame and compensated for gravity.

UndefinedBestEstimate

$\Delta RSDIMU$
$FixedQuantities$
$\Xi Display$
$\Xi Vehicle$

$RSDIMU\_Undefined$
$status' = undefined$
$acceleration' = zero3D$

$$
\begin{array}{|l}
\hline
\textit{ChannelEstimateAcceleration} \\
\Delta RSDIMU \\
\Xi Vehicle \\
\hline
sysStatus \\
\#\{\, f : FaceNames \bullet face(f).status = nonOperational \,\} \leq 2 \\
\forall c : 1 \ldots 4 \bullet channel(i).chanface = channelAssignments(nonOpFaces)(c) \\
\forall c : \mathrm{ran}\ channel \bullet \exists\, acc^I : Vectors3D \bullet \\
\qquad (acc^I, c.status') = getSolution(facePairs(c.chanface)) \wedge \\
\qquad c.acceleration' = transform(acc^I, instrument, navigation, world) + g^N \wedge \\
\qquad c.chanface' = c.chanface \\
world' = world \\
\psi_v' = \psi_v \\
\theta_v' = \theta_v \\
\phi_v' = \phi_v \\
sysStatus' = sysStatus \\
acceleration' = acceleration \\
status' = status \\
\hline
\end{array}
$$

The solution for each channel is arrived at in a similar manner to the best estimate. The only real difference is in the limited set of faces allowed to *getSolution*.

$$
\begin{array}{|l}
\hline
\textit{UndefinedChannelEstimate} \\
\Delta RSDIMU \\
FixedQuantities \\
\Xi Display \\
\Xi Vehicle \\
\hline
\neg\ sysStatus \\
\forall c : \mathrm{ran}\ channel \bullet \\
\qquad c.status' = undefined \wedge \\
\qquad c.acceleration' = zero3D \\
\qquad c.chanface' = 0 \\
world' = world \\
\psi_v' = \psi_v \\
\theta_v' = \theta_v \\
\phi_v' = \phi_v \\
sysStatus = sysStatus' \\
acceleration = acceleration' \\
status = status' \\
\hline
\end{array}
$$

If too many faces are non-operational, the status of the vehicle and the channels is *undefined*, all acceleration values and *chanface* values are set to zero.

Finally, *EstimateAcceleration* is defined by:

$$
\begin{array}{l}
EstimateAcceleration \mathrel{\widehat{=}} (BestEstimateAcceleration \vee UndefinedBestEstimate) \\
\qquad \fatsemi (ChannelEstimateAcceleration \vee UndefinedChannelEstimate)
\end{array}
$$

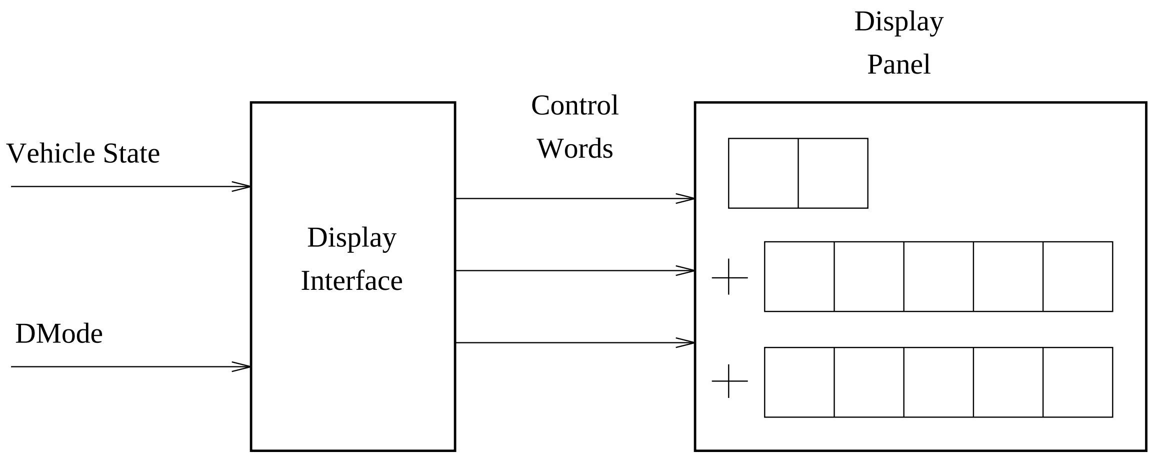

Figure 3: The RSDIMU Display

6 The Display

The display is a component of the RSDIMU that we have largely ignored, to this point. The display presents any of a number of input, intermediate, and output quantities from the calibration and estimation processes depending upon the mode setting of the display.

Figure 3 shows the overall structure of the display. Calibration and estimation information, together with the chosen mode *DMODE* pass through the *DisplayInterface* where the quantities to be displayed are selected and packed into 16-bit words. Individual bits of these words control the lighting of segments on the *DisplayPanel*.

$$
\begin{array}{|l}
ControlWords : \mathbb{P}\ \mathrm{seq}\ \mathbb{Z} \\
\hline
\forall\, w : ControlWords \bullet \\
\mathrm{dom}\, w = 0 \mathrel{..} 15 \wedge \\
\mathrm{ran}\, w \subseteq \{0, 1\}
\end{array}
$$

$$integerEquiv == \lambda\, w : ControlWords \bullet \sum_{i=0}^{15} w(i) * 2^i$$

6.1 The Display Panel

Figure 4 shows the layout of the display panel. The panel is divided into three main areas: the mode indicator, the upper display, and the lower display. The boxes labeled *Mi* and *Di* are seven segment LED (Light-Emitting Diode) displays, whose structure is shown in Figure 5. The *Pi* are decimal points and the *Si* are used to display ± signs.

$$SegmentNames == \{A, B, C, D, E, F, G\}$$
$$segmentOrder == \langle A, B, C, D, E, F, G \rangle$$

$$
\begin{array}{|l}
\hline
\textit{LED} \\
lit : boolean \\
voltage : boolean \\
\hline
\end{array}
$$

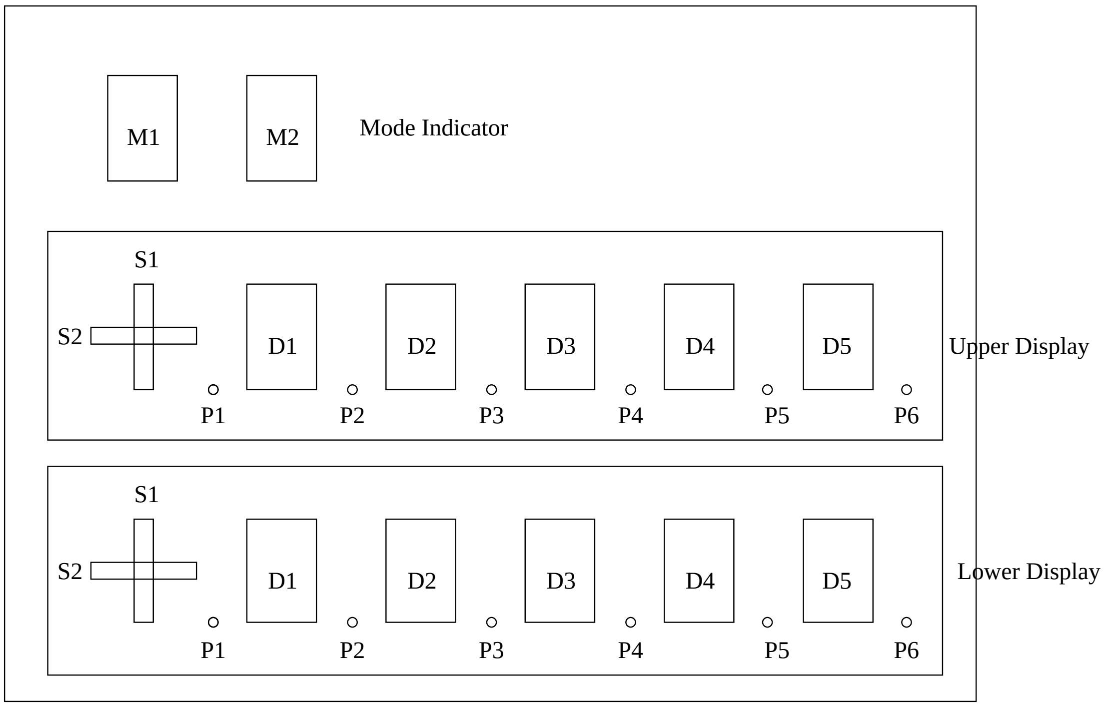

Figure 4: The Display Panel

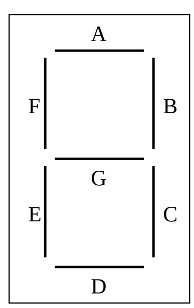

Figure 5: Seven Segment Displays

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{NegativeLogicLED}} \\
\hline
LED \\
\hline
lit = \neg\ voltage \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{PositiveLogicLED}} \\
\hline
LED \\
\hline
lit = voltage \\
\hline
\end{array}
$$

$$
\begin{array}{l}
NegativeLogicLEDs = \{\ NegativeLogicLED\ \} \\
PositiveLogicLEDs = \{\ PositiveLogicLED\ \}
\end{array}
$$

The panel uses a mixture of positive and negative logic LEDs, so that in some cases we must supply a voltage to light a segment, while in other cases the voltage must be supplied to darken a segment.

The set of symbols that can be displayed by a seven segment RSDIMU display are:

$$
\begin{array}{l}
Symbols == \{`0' \mathrel{..} `9', `A' \mathrel{..} `F', `H', `I', `N', `P', `\ '\} \\
symbolOrder == \langle `0', `1', `2', `3', `4', `5', `6', `7', `8', `9', `A', `B', `C', `D', `E', `F' \rangle
\end{array}
$$

A seven segment display can then be described in terms of the segments that must be lit to display each of these symbols:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{SevenSegmentDisplay}} \\
\hline
segment : SegmentNames \rightarrow NegativeLogicLEDs \\
displaying : Symbols \\
\\
litSegments : \mathbb{P}\, SegmentNames \\
\hline
litSegments = \{\ s : SegmentNames \mid segment(s).lit\ \} \\
\\
displaying = `0' \Rightarrow litSegments = \{A, B, C, D, E, F\} \\
displaying = `1' \Rightarrow litSegments = \{B, C\} \\
displaying = `2' \Rightarrow litSegments = \{A, B, D, E, G\} \\
displaying = `3' \Rightarrow litSegments = \{A, B, C, D, G\} \\
displaying = `4' \Rightarrow litSegments = \{B, C, F, G\} \\
displaying = `5' \Rightarrow litSegments = \{A, C, D, F, G\} \\
displaying = `6' \Rightarrow litSegments = \{A, C, D, E, F, G\} \\
displaying = `7' \Rightarrow litSegments = \{A, B, C\} \\
displaying = `8' \Rightarrow litSegments = \{A, B, C, D, E, F, G\} \\
displaying = `9' \Rightarrow litSegments = \{A, B, C, F, G\} \\
displaying = `A' \Rightarrow litSegments = \{A, B, C, E, F, G\} \\
displaying = `B' \Rightarrow litSegments = \{C, D, E, F, G\} \\
displaying = `C' \Rightarrow litSegments = \{A, D, E, F\} \\
displaying = `D' \Rightarrow litSegments = \{B, C, D, E, G\} \\
displaying = `E' \Rightarrow litSegments = \{A, D, E, F, G\} \\
displaying = `F' \Rightarrow litSegments = \{A, E, F, G\} \\
displaying = `H' \Rightarrow litSegments = \{B, C, E, F, G\} \\
displaying = `I' \Rightarrow litSegments = \{E, F\} \\
displaying = `N' \Rightarrow litSegments = \{A, B, C, E, F\} \\
displaying = `P' \Rightarrow litSegments = \{A, B, E, F, G\} \\
displaying = `\ ' \Rightarrow litSegments = \{\} \\
\hline
\end{array}
$$

$$SevenSegmentDisplays == \{\ SevenSegmentDisplay\ \}$$

The seven segment digit display associates a segment name with each of seven LEDs.

The main task in specifying the control panel is simply to indicate which bits in the incoming control words control which segment. Because the upper and lower displays are quite similar in this respect, it is useful to introduce the intermediate concept of a signed 5-digit display.

$$\begin{array}{|l}
\hline \textit{Signed5DigitDisplay} \\
digits : \mathrm{seq}\ SevenSegmentDisplays \\
points : \mathrm{seq}\ PositiveLogicLEDs \\
signs : \mathrm{seq}\ NegativeLogicLEDs \\
words : \mathrm{seq}\ ControlWords \\
\hline
\mathrm{dom}\ digits = 1 \mathrel{..} 5 \\
\mathrm{dom}\ points = 1 \mathrel{..} 6 \\
\mathrm{dom}\ signs = 1 \mathrel{..} 2 \\
\mathrm{dom}\ words = 1 \mathrel{..} 3 \\
\forall i : \mathbb{Z};\ s : SegmentNames \mid i \in 1..7 \land s = segmentOrder(i) \bullet \\
\quad digits(1)(s).voltage = words(1)(i + 6) \land \\
\quad digits(2)(s).voltage = words(1)(i - 1) \land \\
\quad digits(3)(s).voltage = words(2)(i + 6) \land \\
\quad digits(4)(s).voltage = words(2)(i - 1) \land \\
\quad digits(5)(s).voltage = words(3)(i - 1) \\
\forall i : \mathbb{Z} \mid i \in 1..6 \bullet \\
\quad points(i).voltage = words(3)(i + 6) \\
signs(1).voltage = words(3)(14) \\
signs(2).voltage = words(3)(13) \\
\hline
\end{array}$$

$$Signed5DigitDisplays == \{\ Signed5DigitDisplay\ \}$$

For example, the seven segments of the leftmost digit are controlled by bits $7 \mathrel{..} 13$ of the first control word, with segment A controlled by bit 7, segment B by bit 8, and so on.

Similarly, we can describe the mode indicator:

$$\begin{array}{|l}
\hline \textit{ModeIndicator} \\
modeDigits : \mathrm{seq}\ SevenSegmentDisplay \\
modeWord : ControlWords \\
\hline
\mathrm{dom}\ digits = 1 \mathrel{..} 2 \\
\forall i : \mathbb{Z};\ s : SegmentNames \mid i \in 1..7 \land s = segmentOrder(i) \bullet \\
\quad modeDigits(1)(s).voltage = modeWord(1)(i + 6) \land \\
\quad modeDigits(2)(s).voltage = modeWord(1)(i - 1) \\
\hline
\end{array}$$

With these components, we can describe the display panel.

$$\begin{array}{|l}
\hline \textit{DisplayPanel} \\
ModeIndicator \\
upperDisplay : Signed5DigitDisplays \\
lowerDisplay : Signed5DigitDisplays \\
\hline
\end{array}$$

6.2 The Display Interface

The display interface must map various RSDIMU quantities onto the voltages that will provide the desired symbols on the panel.

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{DisplayInterface}} \\
\hline
\textit{DisplayPanel} \\
\textit{DMode} : 0 \mathrel{..} 99 \\
\textit{DisLower}, \textit{DisUpper} : \operatorname{seq} \mathbb{Z} \\
\textit{DisMode} : \mathbb{Z} \\
\hline
\operatorname{dom} \textit{DisLower} = \operatorname{dom} \textit{DisUpper} = 1 \mathrel{..} 3 \\
\forall\, i : \mathbb{Z} \bullet \\
\qquad \textit{DisLower}(i) = \textit{integerEquiv}(\textit{lowerDisplay}.\textit{words}(i)) \land \\
\qquad \textit{DisUpper}(i) = \textit{integerEquiv}(\textit{upperDisplay}.\textit{words}(i)) \\
\textit{DisMode} = \textit{integerEquiv}(\textit{modeWord}) \\
\textit{modeDigits}(1).\textit{displaying} = \textit{symbolOrder}(\textit{DMode}/10) \\
\textit{modeDigits}(2).\textit{displaying} = \textit{symbolOrder}(\textit{DMode} \bmod 10) \\
\hline
\end{array}
$$

The precise mapping between the vehicle state quantities and the panel depends upon the mode *DMode*. In addition, the upper and lower displays can present data in a variety of different formats. These are presented next.

In the "test" format, all LED's are lit:

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{TestFormat}} \\
\hline
\textit{Signed5DigitDisplay} \\
\hline
\forall\, i : \mathbb{Z} \mid i \in 1 \mathrel{..} 5 \bullet \\
\qquad \textit{digits}(i).\textit{displaying} = \text{`8'} \\
\forall\, i : \mathbb{Z} \mid i \in 1 \mathrel{..} 6 \bullet \\
\qquad \textit{points}(i).\textit{lit} \\
\textit{signs}(1).\textit{lit} \\
\textit{signs}(2).\textit{lit} \\
\hline
\end{array}
$$

$$\textit{TestFormat5DigitDisplays} = \{\ \textit{TestFormat}\ \}$$

Similarly, in "blank" format, all LED's must be off

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{BlankFormat}} \\
\hline
\textit{Signed5DigitDisplay} \\
\hline
\forall\, i : \mathbb{Z} \mid i \in 1 \mathrel{..} 5 \bullet \\
\qquad \textit{digits}(i).\textit{displaying} = \text{`\ '} \\
\forall\, i : \mathbb{Z} \mid i \in 1 \mathrel{..} 6 \bullet \\
\qquad \neg\, \textit{points}(i).\textit{lit} \\
\neg\, \textit{signs}(1).\textit{lit} \\
\neg\, \textit{signs}(2).\textit{lit} \\
\hline
\end{array}
$$

$$\textit{BlankFormat5DigitDisplays} = \{\ \textit{BlankFormat}\ \}$$

In "failure" mode, the first digit shows a face name, the second is blank, the third shows the status of the face's x sensor, the fourth is blank, and the fifth shows the status of the face's y sensor.

$$faceToNameMap == \{A \mapsto `A', B \mapsto `B', C \mapsto `C', D \mapsto `D'\}$$

$$\begin{array}{|l}
failureIndicator : Sensors \rightarrow Symbols \\
\hline
\forall s : Sensors \mid \neg\, s.linFail \bullet failureIndicator(s) = `P' \\
\forall s : Sensors \mid s.linNoise \bullet failureIndicator(s) = `N' \\
\forall s : Sensors \mid s.prevFailed \bullet failureIndicator(s) = `I' \\
\forall s : Sensors \mid \neg\,(s.linFail \lor s.linNoise \lor s.prevFailed) \bullet failureIndicator(s) = `F'
\end{array}$$

The function $failureIndicator$ returns 'P' for operational sensors, 'N' for sensors rejected due to excessive noise, 'I' for sensors marked as failed upon input (prior to calibration), and 'F' for sensors that fail during the edge-vector test.

$$\begin{array}{|l}
\multicolumn{1}{l}{FailureFormat} \\
\hline
Signed5DigitDisplay \\
faceToCheck : Faces \\
\hline
digits(1).displaying = faceToNameMap(faceToCheck.sensorFrame) \\
digits(2).displaying = digits(4).displaying = `\ ' \\
digits(3).displaying = failureIndicator(faceToCheck.sensor(x)) \\
digits(5).displaying = failureIndicator(faceToCheck.sensor(y)) \\
\forall i : \mathbb{Z} \mid i \in 1 \mathrel{..} 6 \bullet \neg\, points(i).lit \\
\neg\, signs(1).lit \\
\neg\, signs(2).lit \\
\hline
\end{array}$$

$$FailureFormat5DigitDisplays = \lambda f : Faces \mid f = faceToCheck \bullet FailureFormat$$

In "hexadecimal" format, the display presents an integer quantity as a hexadecimal number.

$$\begin{array}{|l}
\multicolumn{1}{l}{HexFormat} \\
\hline
Signed5DigitDisplay \\
k : \mathbb{Z} \\
\hline
digits(1).displaying = `H' \\
\forall i : \mathbb{Z} \mid i \in 2 \mathrel{..} 5 \bullet \\
\qquad digits(i).displaying = symbolOrder(\left(\frac{k}{16^{5-i}} \bmod 16\right) + 1) \\
\forall i : \mathbb{Z} \mid i \in 1 \mathrel{..} 6 \bullet \neg\, points(i).lit \\
\neg\, signs(1).lit \\
\neg\, signs(2).lit \\
\hline
\end{array}$$

$$HexFormat5DigitDisplays = \lambda i : \mathbb{Z} \mid i = k \bullet HexFormat$$

In "signed decimal" format, the display presents a real number in a fixed point format. The requirements document [1] explicitly specifies several subranges:

$SignedDecimalLow$

$$
\begin{array}{l}
Signed5DigitDisplay \\
r : \Re \\
\hline
r < -99999.0 \\
\\
\neg\ signs(1).lit \\
signs(2).lit \\
\\
points(6).lit \\
\forall\ i : \mathbb{Z} \mid i \in 1 \mathrel{..} 5 \bullet \neg\ points(i).lit \\
\\
\forall\ i : \mathbb{Z} \mid i \in 1 \mathrel{..} 5 \bullet digits(i).displaying = \text{‘9’}
\end{array}
$$

$SignedDecimalHigh$

$$
\begin{array}{l}
Signed5DigitDisplay \\
r : \Re \\
\hline
r > 99999.0 \\
\\
signs(1).lit \\
signs(2).lit \\
\\
points(6).lit \\
\forall\ i : \mathbb{Z} \mid i \in 1 \mathrel{..} 5 \bullet \neg\ points(i).lit \\
\\
\forall\ i : \mathbb{Z} \mid i \in 1 \mathrel{..} 5 \bullet digits(i).displaying = \text{‘9’}
\end{array}
$$

$SignedDecimalNearZero$

$$
\begin{array}{l}
Signed5DigitDisplay \\
r : \Re \\
\hline
-0.000005 < r < 0.000005 \\
\\
\neg\ signs(1).lit \\
\neg\ signs(2).lit \\
\\
points(1).lit \\
\forall\ i : \mathbb{Z} \mid i \in 2 \mathrel{..} 6 \bullet \neg\ points(i).lit \\
\\
\forall\ i : \mathbb{Z} \mid i \in 1 \mathrel{..} 5 \bullet digits(i).displaying = \text{‘0’}
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{SignedDecimalPositive}} \\
\hline
\textit{Signed5DigitDisplay} \\
r : \Re \\
\textit{ptPos} : \mathbb{Z} \\
\textit{norm} : \mathbb{Z} \\
\hline
0.000005 \leq r \leq 99999.0 \\
\textit{signs}(1).\textit{lit} \\
\textit{signs}(2).\textit{lit} \\
\textit{ptPos} = \max(1, 2 + \log_{10} r) \\
\textit{norm} = \textit{trunc}(0.5 + r * (4 - \log_{10} r)) \\
\textit{points}(\textit{ptPos}).\textit{lit} \\
\forall i : \mathbb{Z} \mid i \in 1 \mathinner{.\,.} 6 \land i \neq \textit{ptPos} \bullet \\
\quad \neg\ \textit{points}(i).\textit{lit} \\
\forall i : \mathbb{Z} \mid i \in 1 \mathinner{.\,.} 5 \bullet \\
\quad \textit{digits}(i).\textit{displaying} = \textit{symbolOrder}(\frac{\textit{norm}}{10^{5-i} \bmod 10 + 1}) \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\multicolumn{1}{l}{\textit{SignedDecimalNegative}} \\
\hline
\textit{Signed5DigitDisplay} \\
r : \Re \\
\textit{ptPos} : \mathbb{Z} \\
\textit{norm} : \mathbb{Z} \\
\hline
-99999.0 \leq r \leq -0.000005 \\
\neg\ \textit{signs}(1).\textit{lit} \\
\textit{signs}(2).\textit{lit} \\
\textit{ptPos} = \max(1, 2 + \log_{10}(-r)) \\
\textit{norm} = \textit{trunc}(0.5 + r * (4 - \log_{10}(-r))) \\
\textit{points}(\textit{ptPos}).\textit{lit} \\
\forall i : \mathbb{Z} \mid i \in 1 \mathinner{.\,.} 6 \land i \neq \textit{ptPos} \bullet \\
\quad \neg\ \textit{points}(i).\textit{lit} \\
\forall i : \mathbb{Z} \mid i \in 1 \mathinner{.\,.} 5 \bullet \\
\quad \textit{digits}(i).\textit{displaying} = \textit{symbolOrder}(\frac{\textit{norm}}{10^{5-i} \bmod 10 + 1}) \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textit{SignedDecimalFormat} \mathrel{\widehat{=}} \textit{SignedDecimalLow} \lor \textit{SignedDecimalHigh} \lor \textit{SignedDecimalNearZero} \\
\quad \lor\ \textit{SignedDecimalPositive} \lor \textit{SignedDecimalNegative}
\end{array}
$$

$$
\textit{SignedDecimalFormat5DigitDisplays} = \lambda\, q : \Re \mid r = q \bullet \textit{SignedDecimalFormat}
$$

With the various formats specified, we can now enumerate the various modes of the display interface.

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{TestMode}} \\
\hline
\textit{DisplayInterface} \\
\hline
\textit{DMode} = 88 \\
\textit{upperDisplay} \in \textit{TestFormat5DigitDisplays} \\
\textit{lowerDisplay} \in \textit{TestFormat5DigitDisplays} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{BlankMode}} \\
\hline
\textit{DisplayInterface} \\
\hline
\textit{DMode} \in \{0, 5 \mathrel{..} 20, 25 \mathrel{..} 30, 34 \mathrel{..} 87, 89 \mathrel{..} 99\} \\
\textit{upperDisplay} \in \textit{BlankFormat5DigitDisplays} \\
\textit{lowerDisplay} \in \textit{BlankFormat5DigitDisplays} \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\textit{faceOrders} : \operatorname{seq} \textit{FaceNames} \\
\hline
\textit{faceOrders} = \langle A, B, C, D \rangle
\end{array}
$$

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{FailureMode}} \\
\hline
\textit{DisplayInterface} \\
\textit{faces} : \textit{FaceNames} \rightarrow \textit{Faces} \\
\textit{faceToCheck} : \textit{FaceNames} \\
\hline
\textit{DMode} \in 1 \mathrel{..} 4 \\
\textit{faceOrders}(\textit{faceToCheck}) = \textit{DMode} \\
\\
\textit{upperDisplay} \in \textit{BlankFormat5DigitDisplays} \\
\textit{lowerDisplay} \in \textit{FailureFormat5DigitDisplays}(\textit{faces}(\textit{faceToCheck})) \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{RawlMode}} \\
\hline
\textit{DisplayInterface} \\
\textit{faces} : \textit{FaceNames} \rightarrow \textit{Faces} \\
\textit{faceToCheck} : \textit{FaceNames} \\
\hline
\textit{DMode} \in 21 \mathrel{..} 24 \\
\textit{faceOrders}(\textit{faceToCheck}) = \textit{DMode} + 20 \\
\\
\textit{upperDisplay} \in \textit{HexFormat5DigitDisplays}(\textit{face}(\textit{faceToCheck}).\textit{sensor}(x).\textit{rawl}) \\
\textit{lowerDisplay} \in \textit{HexFormat5DigitDisplays}(\textit{face}(\textit{faceToCheck}).\textit{sensor}(y).\textit{rawl}) \\
\hline
\end{array}
$$

$$
\begin{array}{|l}
\textit{dirOrders} : \operatorname{seq} \textit{DirectionNames} \\
\hline
\textit{dirOrders} = \langle x, y, z \rangle
\end{array}
$$

$$
\begin{array}{|l}
\hline
\textit{BestEstMode} \\
\hline
\textit{DisplayInterface} \\
bestEst : Vectors3D \\
dirToCheck : DirectionNames \\
\hline
DMode \in 31 \mathrel{..} 33 \\
dirOrders(dirToCheck) = DMode + 30 \\
\\
upperDisplay \in SignedDecimalFormat5DigitDisplays(bestEst.x) \\
lowerDisplay \in BlankFormat5DigitDisplays \\
\hline
\end{array}
$$

These modes combine to form the complete specification of the display unit.

$$
\begin{aligned}
Display \mathrel{\widehat=} & \ TestMode \lor BlankMode \lor FailureMode \\
& \lor RawlMode \lor BestEstMode
\end{aligned}
$$

It is worth noting that the entire *Display* has been defined in terms of invariants on the rest of the RSDIMU. There are no state transitions on the display. The display is presumed to continuously show the quantities in the RSDIMUselected by the *DMode*. The requirements further specify that *DMode* is fixed during the period of time covered by this specification (calibration followed by a single in-flight reading).

7 Initialization

Because the requirements for the RSDIMU postulate a driver program that supplies a valid initial state, there is no need for an initialization schema in the usual sense (i.e., for constructing a valid RSDIMU state as a base case). We do, however, need to be concerned with the initialization of quantities introduced in this specification but not explicitly contained in [1]. This initialization can postulate a prior valid state.

$$
\begin{array}{l}
\textit{InitRSDIMU} \\
\hline
\Delta RSDIMU \\
\hline
\forall f : FaceNames;\ d : SensorNames \bullet \\
\qquad face(f).sensor(d).prevFailed' = face(f).sensor(d).linFail \qquad (5) \\
\\
world' = world \\
\psi'_I = \psi_I \\
\theta'_I = \theta_I \\
\phi'_I = \phi_I \\
obase' = obase \\
\\
\forall f : FaceNames \bullet \\
\qquad (face(f).sensorFrame' = face(f).sensorFrame \wedge \\
\qquad face(f).misalign' = face(f).misalign \wedge \\
\qquad face(f).temp' = face(f).temp \wedge \\
\qquad face(f).normFace' = face(f).normFace) \\
\\
\forall f : FaceNames;\ d : SensorNames \bullet \\
\qquad (face(f).offRaw' = face(f).offRaw \wedge \\
\qquad face(f).sensor(d).scale0' = face(f).sensor(d).scale0 \wedge \\
\qquad face(f).sensor(d).scale1' = face(f).sensor(d).scale1 \wedge \\
\qquad face(f).sensor(d).scale2' = face(f).sensor(d).scale2 \wedge \\
\qquad face(f).sensor(d).rawl' = face(f).sensor(d).rawl \wedge \\
\qquad face(f).sensor(d).linNoise' = face(f).sensor(d).linNoise \wedge \\
\qquad face(f).sensor(d).linOffset' = face(f).sensor(d).linOffset \wedge \\
\qquad face(f).sensor(d).linFail' = face(f).sensor(d).linFail) \\
\\
DMode' = DMode \\
\hline
\end{array}
$$

The only real point of interest here is equation 5, which indicates that *prevFailed* holds the sensor failure information prior to our beginning any calibration activities. (It corresponds therefore to the value of the variable `LINFAILIN` in [1].

8 Conclusions

The RSDIMU project was selected to give students experience in the process of writing formal specifications for a realistic project. Without this experience, students are often inclined to dismiss formal methods as having low relevance outside of academia. The small examples that fill most textbooks, useful as they are for pedagogical purposes, do little to dispell this prejudice. Even the longer case studies from [2] are insufficiently convincing. For example, none of the case studies in [2] illustrate the use of multiple layers of Z-defined abstract objects (except in one-to-one containment), which proved so important in this specification.

The RSDIMU requirements were useful because they were clearly authentic, and because they were involved enough to be quite intimidating at the start of the project. Student comments at the end of the semester indicated that they were, in fact, surprised at how much they had been able to accomplish in a relatively short period of time.

References

[1] Charles Rivers Analytics, Inc. Redundancy management software: Requirements specification for a redundant strapped down inertial measurement unit. Technical Report NASA.FTS87/RSDIMU/RS/3.2/Feb-87, NASA, Feb. 1987. Version 3.2.

[2] A. Diller. *Z – an Introduction to Formal Methods*. John Wiley & Sons, 1990.

[3] D. E. Eckhardt, A. K. Caglayan, J. C. Knight, L. D. Lee, D. F. McAllister, M. A. Vouk, and J. P. J. Kelly. An experimental evaluation of software redundancy as a strategy for improving reliability. *IEEE Transactions on Software Engineering*, 17(7):692–702, July 1991.

[4] P. R. Lorczak and A. K. Caglayan. A large scale second generation experiment in multi-version software: Analysis of software and specification errors. Technical Report R8903, Charles River Analytics, Inc., Jan. 1989.

Index