

SOFTWARE REQUIREMENTS

Guidance and Control Software Development Specification

Version 2.2 with formal mods 1-26

June 7, 1993

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681

Preface

The NASA Langley Research Center has been conducting a series of software error studies in an effort to better understand the software failure process and improve development and reliability estimation techniques for avionics software. The Guidance and Control Software (GCS) project is the latest study in the series. This project involves production of guidance and control software for the purpose of gathering failure data from a credible software development environment. To increase the credibility and relevance of this study, guidelines used in the development of commercial aircraft were adopted. The use of the Radio Technical Commission for Aeronautics RTCA/DO-178A guidelines, "Software Considerations in Airborne Systems and Equipment Certification," is required by the Federal Aviation Administration (FAA) for developing software to be certified for use in commercial aircraft equipment [1].

This is document #2 in the series of documents required to fulfill the RTCA/DO-178A guidelines. The documents in the series are numbered as specified in the DO-178A guidelines and are used to demonstrate compliance with the guidelines by describing the application of the procedures and techniques used during the development of flight software. For the GCS project, the series consists of the following documents:

- *GCS Configuration Index* Document no. 1
- *GCS Development Specification* Document no. 2
- *GCS Design Description* Document no. 3
- *GCS Programmer's Manual* Document no. 4 (this document includes *Software Design Standards* Document no. 12)
- *GCS Configuration Management Plan* Document no. 5A
- *Software Quality Assurance Plan for GCS* Document no. 5B
- *GCS Source Listing* Document no. 6
- *GCS Source Code* Document no. 7
- *GCS Executable Object Code* Document no. 8 (not available in hardcopy)
- *GCS Support/Development System Configuration Description* Document no. 9

- *GCS Accomplishment Summary* Document no. 10
- *Software Verification Plan for GCS* Document no. 11
- *GCS Development Specification Review Description* Document no. 11A
- *GCS Simulator (GCS_SIM) System Description* Document no. 13
- *GCS Simulator (GCS_SIM) Certification Plan* Document no. 13A
- *GCS Plan for Software Aspects of Certification* Document no. 14

A GCS implementation (code which fulfills the requirements outlined in the *Guidance and Control Software Development Specification*) runs in conjunction with a software simulator that provides input based on an expected usage distribution in the operational environment, provides response modeling, and receives data from the implementation. For the purposes of the project, a number of GCS implementations are being developed by different programmers according to the structured approach found in the DO-178A guidelines. The GCS simulator is designed to allow an experimenter to run one or more implementations in a multitasking environment and collect data on the comparison of the results from multiple implementations. Certain constraints have been incorporated in the software requirements due to the nature of the GCS project. Further information on goals of the GCS project are available in the *GCS Plan for Software Aspects of Certification*.

FOREWORD

This specification defines a guidance and control system for a planetary landing vehicle **during its terminal phase of descent**. **The guidance and control system** is specified using an extension to the popular method of structured analysis. This specification is written for an experienced programmer with two or more years of full-time industrial programming experience using a scientific programming language. The programmer should have an adequate background, either through college courses or job training in mathematics, physics, differential equations, and numerical integration. **The specification was written with the assumption that the implementation would be coded in FORTRAN; however, other languages can be used.**

Version 2.2 of this specification contains a number of modifications to version 2.1 of the specification. The text that has been modified from version 2.1 is bolded in version 2.2. Some existing text has been moved to another place in the document, and some text has been deleted. There is no demarcation to indicate where text has been moved or deleted. The modifications that are significant (may impact the coding of an implementation) are marked with a footnote number (note that there is only a footnote number and not a traditional footnote containing an explanation of the change). If there are a number of significant modifications within a processing step (in Chapter 5 of the specification), a footnote number has been placed just at the top of the processing step (as opposed to marking each individual change within the processing step). Note that there is a significant new addition to the specification: requirements for exception handling. New additions to the text are also bolded.

Contents

PREFACE	i
FOREWORD	iii
1. INTRODUCTION	1
INTRODUCTION	3
PURPOSE OF THE GUIDANCE AND CONTROL SOFTWARE.....	3
VEHICLE CONFIGURATION.....	3
TERMINAL DESCENT	6
VEHICLE DYNAMICS	6
Frames of Reference.....	6
Linear Velocity.....	7
Vehicle Position	7
Angular Velocity	8
Vehicle Attitude.....	8
Acceleration.....	8
Further Reading.....	8
VEHICLE GUIDANCE.....	10
ENGINES	10
Axial Engine (Thrust) Control.....	10
Roll Engine Control.....	10
GENERAL INFORMATION	11
NOTATION.....	11
Matrices and Arrays.....	11
Operators	11
DEFINITIONS	12
CONVENTIONS.....	14
FORTRAN Convention.....	14
REQUIREMENTS	14
Order of Processing.....	14
Calls to GCS_SIM_RENDEZVOUS.....	14
Control Signals	14
Number Representations.....	14
Conversion of Units.....	14
Global Data Store Organization	14
Use of Variables That Are Not in the Global Data Stores.....	15

Use of Tables.....	15
Rotation of History Variables.....	15
EXCEPTION HANDLING.....	15
Exception Conditions.....	16
Action to be Taken for Each Specified Exception Condition.....	16
Output to be Generated for Each Exception Condition.....	16
2. LEVELS 0 AND 1 SPECIFICATION.....	19
LEVEL 0 SPECIFICATION.....	21
LEVEL 1 SPECIFICATION.....	26
3. LEVEL 2 SPECIFICATION.....	29
PROCESS SPECIFICATION (P-Spec) 1: INIT_GCS.....	31
4. LEVEL 3 FLOW DIAGRAMS AND C-SPECS.....	35
SCHEDULING.....	43
5. P-SPECS FOR LEVELS 3 AND 4.....	45
AECLP -- Axial Engine Control Law Processing (P-Spec 2.3.1).....	47
ARSP -- Altimeter Radar Sensor Processing (P-Spec 2.1.2).....	53
ASP -- Accelerometer Sensor Processing (P-Spec 2.1.1).....	55
CP -- Communications Processing (P-Spec 2.4).....	59
CRCP -- Chute Release Control Processing (P-Spec 2.3.3).....	63
GP -- Guidance Processing (P-Spec 2.2).....	65
GSP -- Gyroscope Sensor Processing (P-Spec 2.1.4).....	73
RECLP -- Roll Engine Control Law Processing (P-Spec 2.3.2).....	75
TDLRSP -- Touch Down Landing Radar Sensor Processing (P-Spec 2.1.3).....	77
TDSP -- Touch Down Sensor Processing (P-Spec 2.1.6).....	83
TSP -- Temperature Sensor Processing (P-Spec 2.1.5).....	85
6. DATA REQUIREMENTS DICTIONARY.....	89
PART I. DATA ELEMENT DESCRIPTIONS.....	91
PART II. CONTENTS OF DATA STORES.....	107
PART III. CONTROL SIGNALS, DATA CONDITIONS, AND GROUP FLOWS.....	111
A. NOTATION FOR LEVELS 0, 1, 2, AND 3 SPECIFICATION.....	115
B. IMPLEMENTATION NOTES.....	119
INTERFACE.....	121
Background.....	121
Simulator Support Utility.....	121
Input/Output.....	121
Process.....	123

GCS Initialization	123
C. NUMERICAL INTEGRATION INSTRUCTIONS.....	125
BIBLIOGRAPHY	129

List of Figures

1.1 THE LANDING VEHICLE DURING DESCENT	4
1.2 A TYPICAL TERMINAL DESCENT TRAJECTORY	5
1.3 ENGINEERING ILLUSTRATION OF VEHICLE.....	9
2.1 STRUCTURE OF THE GCS SPECIFICATION	23
2.2 DATA CONTEXT DIAGRAM: LANDER.....	24
2.3 CONTROL CONTEXT DIAGRAM: LANDER.....	25
2.4 DATA FLOW DIAGRAM (DFD) 0: GCS.....	26
2.5 CONTROL FLOW DIAGRAM (CFD) 0: GCS	27
3.1 DFD 2: RUN_GCS	32
3.2 CFD 2: RUN_GCS	33
4.1 DFD 2.1: SP -- SENSOR PROCESSING.....	37
4.2 CFD 2.1: SP -- SENSOR PROCESSING.....	38
4.3 DFD 2.3: CLP -- CONTROL LAW PROCESSING	40
4.4 CFD 2.3: CLP -- CONTROL LAW PROCESSING.....	41
5.1 VELOCITY-ALTITUDE CONTOUR	69
5.2 GRAPH FOR DERIVING ROLL ENGINE COMMANDS.....	76
5.3 DOPPLER RADAR BEAM LOCATIONS.....	78
5.4 CALIBRATION OF THERMOCOUPLE PAIR.....	86
A.1 GRAPHICAL SYMBOLS USED IN FLOW DIAGRAMS	118
B.1 DIAGRAM OF STORAGE AS SEEN BY GCS IMPLEMENTATIONS	122

List of Tables

1.1 ROTATION OF VARIABLES.....	15
2.1 CONTROL SPECIFICATION (C-SPEC) 0: GCS	28
3.1 C-Spec 2: RUN_GCS	34
4.1 C-Spec 2.1: SP -- Sensor Processing	39
4.2 C-Spec 2.3: CLP -- Control Law Processing	42
4.3 FUNCTIONAL UNIT SCHEDULING	43
5.1 DETERMINATION OF AXIAL ENGINE TEMPERATURE	48
5.2 DETERMINATION OF ERROR TERMS.....	50
5.3 DETERMINATION OF AXIAL ENGINE COMMANDS.....	50
5.4 DETERMINATION OF ALTITUDE STATUS.....	54
5.5 PACKET VARIABLES.....	61
5.6 SAMPLE MASK	61
5.7 EXAMPLE OF PACKET	62
5.8 DIFFERENTIAL EQUATIONS	66
5.9 DETERMINATION OF AXIAL AND ROLL ENGINE ON/OFF SWITCHES.....	67
5.10 DETERMINATION OF GUIDANCE PHASE	70
5.11 DETERMINATION OF RADAR BEAM STATES	79
5.12 PROCESSING OF DOPPLER RADAR BEAMS IN LOCK.....	81
5.13 DETERMINATION OF TOUCH DOWN SENSOR AND STATUS	83
6.1 DATA STORE: GUIDANCE_STATE	107
6.2 DATA STORE: EXTERNAL.....	108
6.3 DATA STORE: SENSOR_OUTPUT.....	108
6.4 DATA STORE: RUN_PARAMETERS.....	109
6.5 CONTROL VARIABLES	111
6.6 DATA CONDITIONS.....	111
6.7 INITIALIZATION DATA.....	112
6.8 TEMP_DATA	114
6.9 SENSOR_DATA	114
6.10 OUTPUT_DATA	114
6.11 OUTPUT_CONTROL.....	114
6.12 FRAME_DATA.....	114
C.1 INITIAL VALUES PROVIDED FOR USE IN INTEGRATION	128

1. INTRODUCTION

INTRODUCTION

PURPOSE OF THE GUIDANCE AND CONTROL SOFTWARE

The **Guidance and Control Software (GCS)** represents the **Viking lander on-board navigational software**. The purpose of this software is to:

1. provide guidance and engine control of the vehicle (shown in Figure 1.1) during its terminal phase of descent onto a surface and
2. communicate sensory information about the vehicle and its descent to some other receiving device.

A typical descent trajectory is shown in Figure 1.2.

The initialization of the GCS starts the sensing of vehicle altitude. When a predefined engine ignition altitude is sensed by the altimeter radar, the GCS begins guidance and control of the vehicle. The axial and roll engines are ignited; while the axial engines are warming up, the parachute remains connected to the vehicle. During this engine warm-up phase, the aerodynamics of the parachute dictate the trajectory followed by the vehicle. Vehicle attitude is maintained by firing the engines in a throttled-down condition. Once the main engines become hot, the parachute is released and the GCS **performs an attitude correction maneuver and then follows a controlled acceleration descent until a predetermined velocity-altitude contour is crossed (see Figure 5.1)**. The GCS then attempts to maintain the descent of the vehicle along this predetermined velocity-altitude contour. The vehicle descends along this contour until a predefined engine shut off altitude is reached or touchdown is sensed. After all engines are shut off, the vehicle free-falls to the surface.

VEHICLE CONFIGURATION

The vehicle to be controlled is a guidance package containing sensors which obtain information about the vehicle state, a guidance and control computer, and actuators providing the thrust necessary for maintaining a safe descent. The vehicle has three accelerometers (one for each body axis), one doppler radar with four beams, one altimeter radar, two temperature sensors, three strapped-down gyroscopes, three opposed pairs of roll engines, three axial thrust engines, one parachute release actuator, and a touch down sensor. The vehicle has a hexagonal, box-like shape with three legs and a surface sensing rod protruding from its undersurface.

Figure 1.1: THE LANDING VEHICLE DURING DESCENT¹

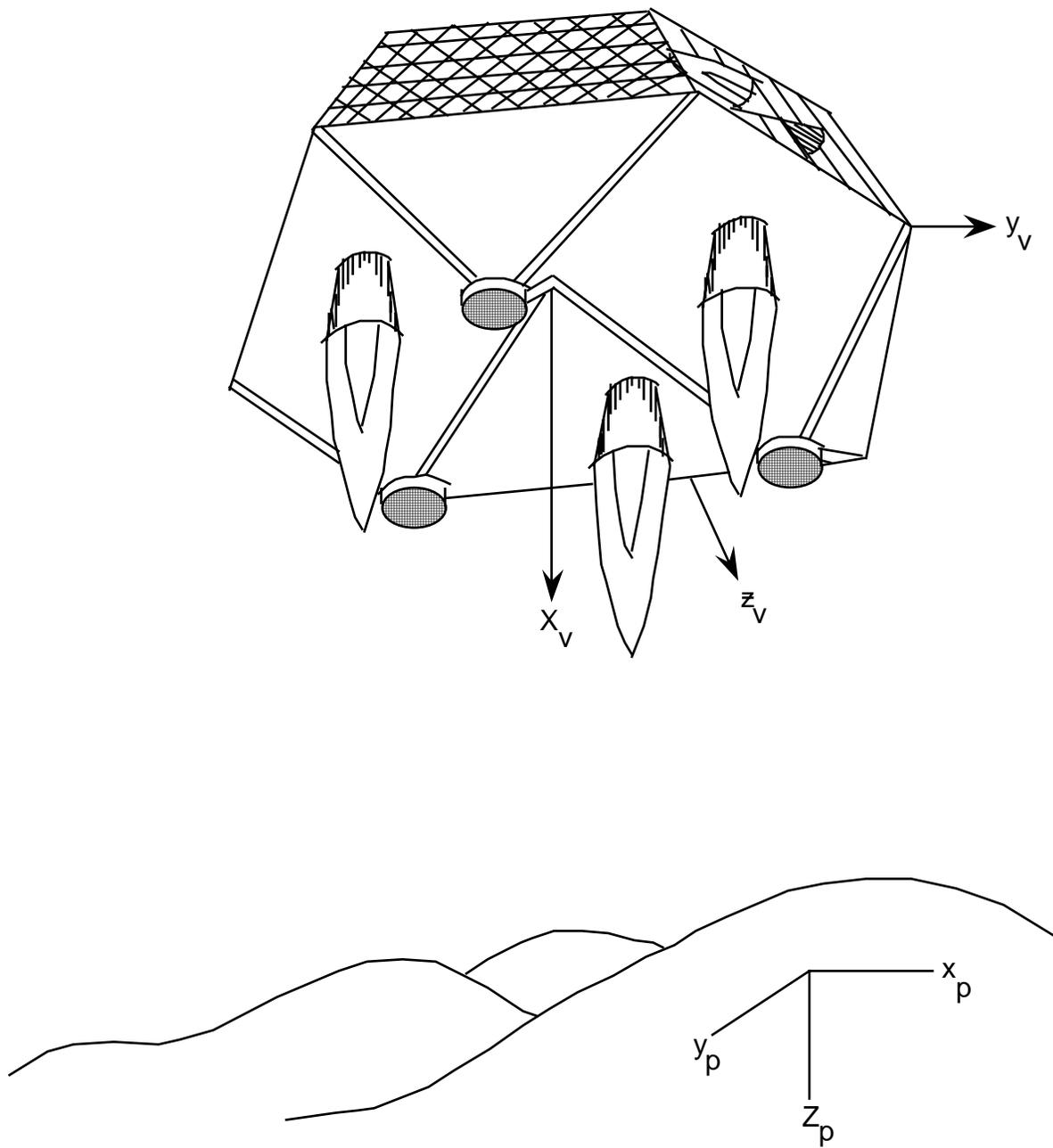
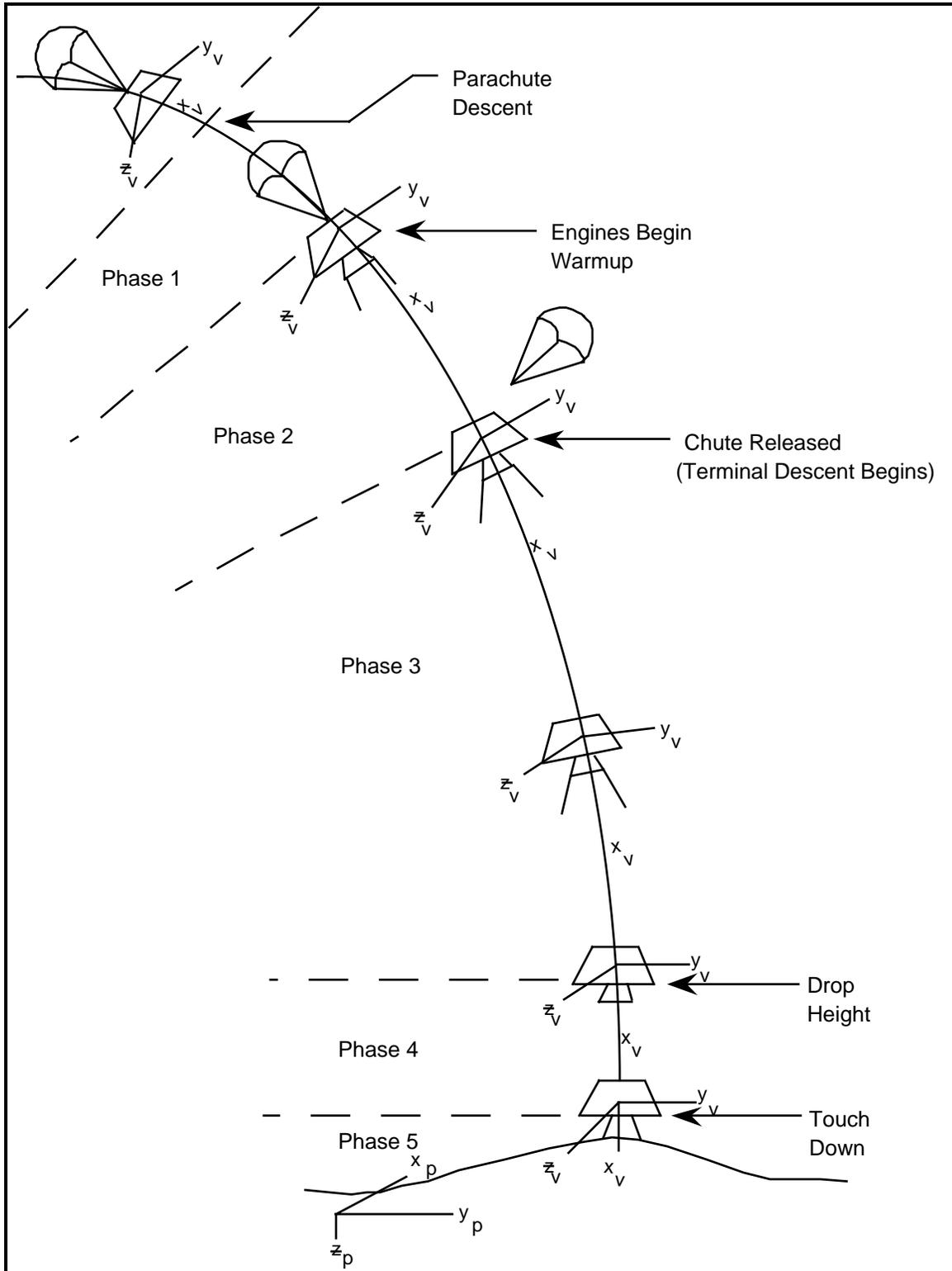


Figure 1.2: A TYPICAL TERMINAL DESCENT TRAJECTORY³



TERMINAL DESCENT

Prior to the terminal descent phase, the vehicle falls with a parachute attached. This parachute is released seconds after the engines ignite, and terminal descent begins. During terminal descent, the vehicle follows a modified gravity-turn guidance law until a predetermined altitude is reached. The atmosphere introduces drag forces, including the random effects of wind. **Independently** throttled engines slow the vehicle down. These engines can control the vehicle's orientation, and roll engines control the vehicle's roll rate. Roll control is necessary to keep the doppler radars in lock and insure that the desired touch down attitude (land on two legs prior to the third) is maintained.

The velocity during descent follows the predetermined velocity-altitude contour. At a **specific altitude** above the planet surface, the vehicle is maintained at a constant descent velocity. Once the surface is sensed, all engines are shut down and the vehicle free falls to the surface.

VEHICLE DYNAMICS

Frames of Reference

Terminal descent is described in terms of two coordinate systems:

1. the surface-oriented coordinate system, and
2. the vehicle-oriented coordinate system.

In the surface coordinate system, the \vec{z}_p axis is viewed as normal to the surface and points down as shown in Figure 1.2. The \vec{x}_p axis points north, and the \vec{y}_p points east.

By defining a *unit vector* as a vector of length equal to one unit along each axis in both the planetary and vehicular frames of reference, a relation between these two frames of reference may be established. Any vector can then be defined as a multiple of the unit vector along each of the axes defined in the frame of reference. Thus, the velocity of the vehicle \vec{V} may be defined in the vehicle's frame of reference as: $V_{x_v} \hat{i}_v + V_{y_v} \hat{j}_v + V_{z_v} \hat{k}_v$, where \hat{i}_v , \hat{j}_v , and \hat{k}_v are the unit vectors in the x , y , and z directions of the vehicles coordinate system (unit vectors are usually represented by lower case i , j , or k with a hat to show that they are unit vectors). V_{x_v} , V_{y_v} , and V_{z_v} represent the components of the vehicle velocity in the given direction. At the same time, the velocity of the vehicle may be described in the planetary coordinate system as: $V_{x_p} \hat{i}_p + V_{y_p} \hat{j}_p + V_{z_p} \hat{k}_p$, where the subscript p represents planetary rather than vehicle coordinates. Note, since the two coordinate systems are not oriented in the same direction, the values of V_{x_v} will not be equal to V_{x_p} , but the magnitude of the total vector \vec{V} will be the same in both systems. Also the difference in the magnitudes of individual components represents the difference in relative orientation between the two coordinate systems.

The *dot product* ($\vec{a} \cdot \vec{b}$) is defined as the magnitude of \vec{a} multiplied by the magnitude of \vec{b} and then by the cosine of the angle between the vectors,

$$\vec{a} \cdot \vec{b} = |\vec{a}| |\vec{b}| \cos \angle ab$$

The dot product is used to project \vec{a} onto \vec{b} and can be used to project a vector in one frame of reference onto another one. Rather than calculate the needed cosines each time a vector must be transformed from one frame of reference into another, the cosines of the angles between each unit vector of the vehicular and planetary coordinate systems are computed and placed into a *direction cosine matrix*. This matrix is then used along with the vector's magnitude in each dimension of the original frame of reference to compute a dot product. This product gives the vector's magnitude in each dimension of the new frame of reference.

The transformation between the vehicle and the surface coordinate systems at time t is specified by a matrix of direction cosines,

$$\begin{pmatrix} l_1 & l_2 & l_3 \\ m_1 & m_2 & m_3 \\ n_1 & n_2 & n_3 \end{pmatrix}_t = \begin{pmatrix} \cos \theta(\hat{i}_v, \hat{i}_p) & \cos \theta(\hat{i}_v, \hat{j}_p) & \cos \theta(\hat{i}_v, \hat{k}_p) \\ \cos \theta(\hat{j}_v, \hat{i}_p) & \cos \theta(\hat{j}_v, \hat{j}_p) & \cos \theta(\hat{j}_v, \hat{k}_p) \\ \cos \theta(\hat{k}_v, \hat{i}_p) & \cos \theta(\hat{k}_v, \hat{j}_p) & \cos \theta(\hat{k}_v, \hat{k}_p) \end{pmatrix}_t$$

where $\theta(\hat{i}, \hat{j})$ denotes the angle between vectors \hat{i} and \hat{j} , etc.

The change in orientation of the vehicle during descent makes the update of the direction cosine matrix necessary at each time step. This update is specified in the following equation:

$$d/dt \begin{pmatrix} l_1 & l_2 & l_3 \\ m_1 & m_2 & m_3 \\ n_1 & n_2 & n_3 \end{pmatrix}_t = \begin{pmatrix} 0 & r_v & -q_v \\ -r_v & 0 & p_v \\ q_v & -p_v & 0 \end{pmatrix}_t \begin{pmatrix} l_1 & l_2 & l_3 \\ m_1 & m_2 & m_3 \\ n_1 & n_2 & n_3 \end{pmatrix}_t$$

where the matrix containing the p_v , q_v , and r_v terms is the rate of rotation about the axes of the vehicle which may be obtained from sensor values.

Linear Velocity

The linear components of velocity for the vehicle during terminal descent are denoted by \dot{X}_v , \dot{Y}_v , and \dot{Z}_v in the vehicle coordinate system and by \dot{X}_p , \dot{Y}_p , and \dot{Z}_p in the surface coordinate system, where the dot ($\dot{}$) notation indicates derivatives with respect to time.

Vehicle Position

Vehicle position is expressed in terms of the surface coordinate system by transforming change in position (velocity) in the vehicle coordinate system into change in position in the surface frame and integrating as follows:

$$\begin{pmatrix} \dot{x}_p \\ \dot{y}_p \\ \dot{z}_p \end{pmatrix}_t = \begin{pmatrix} l_1 & m_1 & n_1 \\ l_2 & m_2 & n_2 \\ l_3 & m_3 & n_3 \end{pmatrix}_t \begin{pmatrix} \dot{x}_v \\ \dot{y}_v \\ \dot{z}_v \end{pmatrix}_t$$

and

$$\begin{pmatrix} x_p \\ y_p \\ z_p \end{pmatrix}_t = \int \begin{pmatrix} \dot{x}_p \\ \dot{y}_p \\ \dot{z}_p \end{pmatrix} d\tau \Big|_t$$

Angular Velocity

Roll, pitch, and yaw angular velocities are represented by the quantities p_v , q_v , and r_v in the vehicle frame of reference only. Roll is about the \vec{x}_v axis, pitch is about the \vec{y}_v axis, and yaw is about the \vec{z}_v axis, as shown in Figure 1.3. A more in-depth explanation of angular velocity naming conventions and other related material may be found in section II, part B of Reference [3].

Vehicle Attitude

The vehicle attitude at time t is a function of the vehicle attitude (known by reference to celestial objects) at the start of descent at time t_0 and the cumulative changes in attitude from time t_0 to time t .

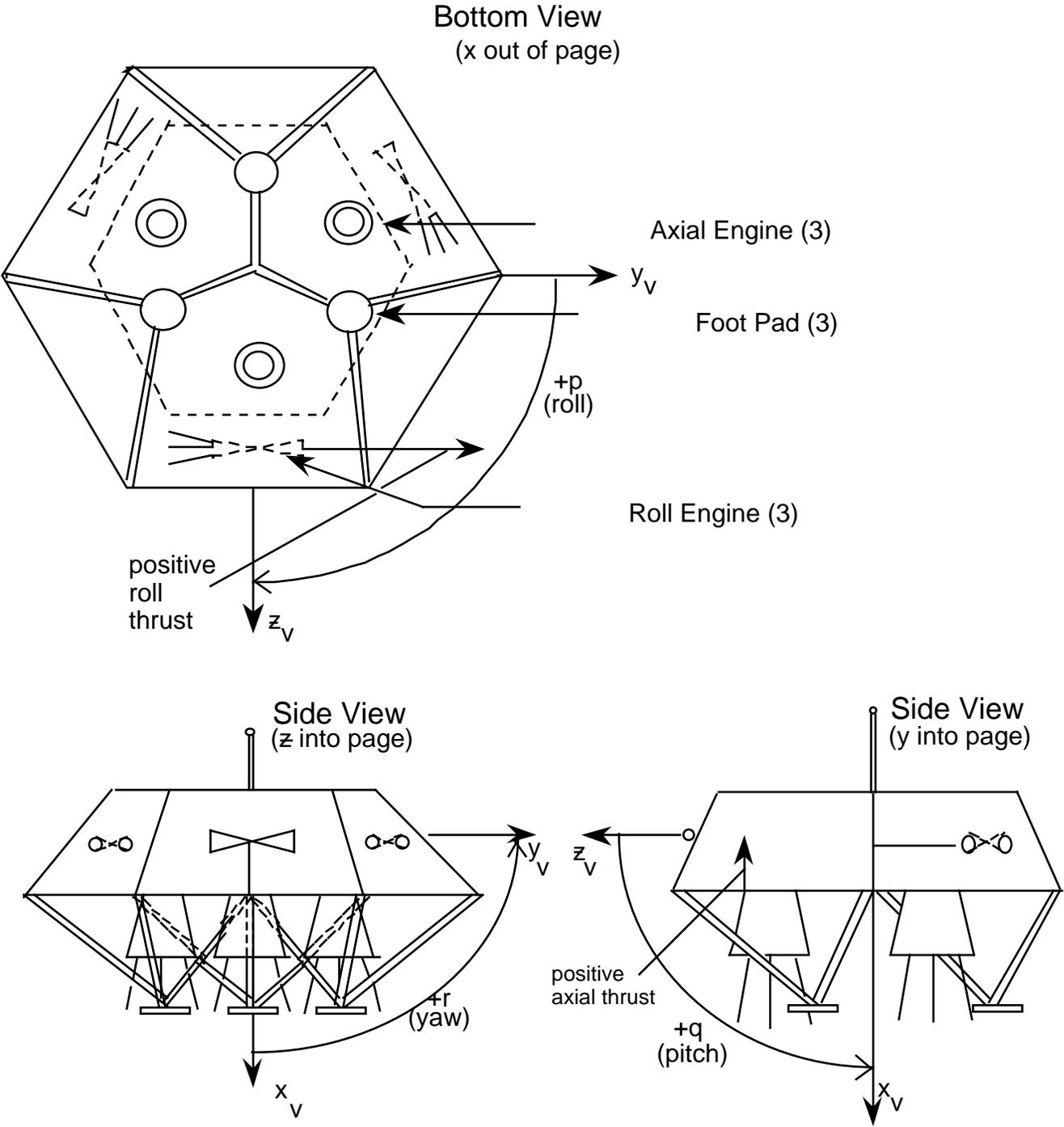
Acceleration

The linear components of acceleration for the vehicle in the vehicle frame of reference during terminal descent are denoted by \mathbb{X}_v , \mathbb{Y}_v , and \mathbb{Z}_v , respectively.

Further Reading

The subjects of vector mathematics, transformations between frames of references, vector calculus, and rotating coordinate systems may not be sufficiently covered here for the user; however, such depth is not intended for this document. Chapter 4 of *Classical Mechanics* [4] contains a detailed explanation of rigid body motion and transformation of vectors into multiple frames of reference or coordinate systems. Chapters 15 and 16 of *Engineering Mechanics* [5] contains a more basic approach to the same ideas of multiple frames of reference and vector mechanics. Chapter 14 of [6] and Chapter 5 of [7] also discuss rotational motion and multiple frames of reference, as well as vector mechanics and calculus. Two other books of possible interest are [8] and [9]. Both cover the mechanics of particles and dynamics, with strong references to particle trajectories and rocket dynamics. Also, these texts are basic in nature and require only a rudimentary knowledge of physics, math, or engineering.

Figure 1.3: ENGINEERING ILLUSTRATION OF VEHICLE⁵



VEHICLE GUIDANCE

Vehicle guidance is accomplished by varying the engine thrust so that the vehicle follows a single predetermined velocity-altitude contour. This contour is made available during GCS initialization. Applying too great a deceleration early in the descent brings the vehicle velocity to its terminal value too high above the surface, resulting in insufficient propellant for final descent. Applying too small a thrust lets the vehicle impact the surface with too great a velocity. Either condition could be disastrous. As soon as the touch down sensor touches the surface, the engines are shut off. Approximately ninety percent of propellant or thrust is used to minimize gravity losses; the remaining ten percent is used for steering.

A gravity-turn steering law is mechanized by rotating the vehicle in pitch and yaw until the body's lateral axis velocities are zero (causing the thrust axis to point along the total velocity vector). The action of gravity causes the thrust axis to rotate toward the vertical as the total velocity is reduced. An arbitrary roll orientation is maintained with an attitude hold mode during the descent.

ENGINES

The vehicle has three axial engines that supply the force necessary to slow the vehicle and allow it to safely land. Roll is controlled by three pairs of roll engines on the lander supplying rotational thrust. Figure 1.3 shows the axial and roll engines and the resulting thrust forces they impart to the vehicle.

Axial Engine (Thrust) Control

Three thrust engines first orient the vehicle so that their combined thrust vector opposes the vehicle's velocity vector. Thrust (axial direction) engine control is a function of pitch error, yaw error, thrust error, and deviation from the velocity-altitude contour. A combination of proportional and integral control (PI) logic is applied to pitch and yaw control. The integral portion helps to reduce the steady-state pitch and yaw error.

If no thrust error or velocity-altitude contour deviation occurs, then axial engine response provides only pitch and yaw control via the PI control law. Use of this control law implies that the overshoot problem for pitch-yaw control is probably small.

Thrust control is implemented by a proportional-integral-derivative (PID) control law. The derivative control added here damps out overshoot.

Roll Engine Control

Roll control is attained by pulsing the three pairs of roll engines and is a function of roll angle deviation and roll rate (p_V) about the x axis. Roll engine specific impulse and thrust per unit time are constant with the integrated thrust controlled by pulse rate. Angle deviations are controlled within a very small range of 0.25 to 0.35 degrees.

GENERAL INFORMATION

NOTATION

Matrices and Arrays

It should be noted that throughout this specification, the words matrix and array are often interchanged. No significance should be placed upon the use of one word as opposed to use of the other.

All matrices are referenced with the row index first and the column index second. In the cases where there is a time history (see definition of history variable below), the last index is the time index.

When the name of an array which contains a time history is given without any indices being specified, the most recent value is implied.

Operators

Throughout this specification, matrix operations (particularly multiplication) are required, and on some occasions, non-standard operations are used upon matrices. The following symbols are used to denote the types of multiplication to be applied.

Dots (·) Small dots are used to denote scalar multiplication. For example:

$$3 \cdot 4 = 12$$

Multiplication sign (×) This symbol is used to denote standard matrix multiplication. This does NOT imply a cross product, nor strictly a dot product. The definition of this type of operation is given below:

$$A \times B = C$$

where

$$C_{ij} = \sum_{k=1}^n A_{ik} \cdot B_{kj}$$

Asterisks (*) Asterisks are used in conjunction with index markers to show that the operations are to be conducted on individual elements of arrays or vectors as if they were scalars. This is often the case when calculating sensor values or other similar functions when multiple scalars are grouped together for convenience. For example, the following equation is listed in ASP:

The equation for measured acceleration is:

$$A_ACCELERATION_M(i) = A_BIAS(i) + A_GAIN(i) * A_COUNTER(i)$$

where i ranges from 1 to 3 and represents the three directions x, y, and z. In this case, the first element of $A_ACCELERATION_M$ would be calculated as follows:

$$A_ACCELERATION_M(1) = A_BIAS(1) + A_GAIN(1) \cdot A_COUNTER(1)$$

No Operator In those cases where variables, matrices, or scalars are located directly beside each other with no operator between, standard multiplication is implied. Thus two matrices collocated would be multiplied as if they had the \times operator between them, while two scalars would be multiplied as if they had the \cdot operator between them. Also, if a scalar and a matrix (of one or more dimensions) were collocated, then the scalar would be multiplied by each element of the matrix and a new matrix of equal dimensions would be generated.

DEFINITIONS

Implementation

Computer code which fulfills all of the requirements outlined in the GCS Development Specification.

Functional Unit

Chapter 5 is divided into eleven subsections, each of which describes the requirements for a particular function to be performed by the GCS software. Throughout this specification, the term "functional unit" will be used to refer to one of these eleven functions. Note that there is not necessarily a one-to-one correspondence between a "functional unit" and a distinct unit or module of software code in an implementation.

Frame

A frame is the length of time necessary to execute all scheduled functional units. Each frame has two different time values associated with it. The first is the actual c.p.u. time that it takes to execute the GCS software on the simulation host computer, while the second is the allotted time for a frame on the actual lander. The global variable $DELTA_T$ represents the time for one frame on the actual lander and is needed in the GCS code for the integration of the dynamic equations for the lander.

Subframe

A subframe is one of the three individual units of time which together make up a frame. The three subframes are named the Sensor Processing subframe (subframe 1), the Guidance Processing subframe (subframe 2), and the Control Law Processing subframe (subframe 3). In each frame, subframe 1 is executed first, subframe 2 is executed second, and subframe 3 is the last subframe executed.

Data Store

The definition for a data or control store given in Hatley [13] is "A data or control store is simply a data or control flow frozen in time. The data or control information it contains may be used any time after that information is stored and in any order." In this specification, all stores contain data, while some also contain data conditions. For the purposes of this specification, the term "data store" will be used to refer to any store which contains some combination of data and data conditions. Thus, all four stores listed in the Data Requirements Dictionary part II will be referred to as "data stores".

Global Data Store Variable

A global data store variable is any variable listed in any of the four global data stores in Chapter 6, namely GUIDANCE_STATE data store (Table 6.1), EXTERNAL data store (Table 6.2), SENSOR_OUTPUT data store (Table 6.3), or RUN_PARAMETERS data store (Table 6.4).

History Variable

Within this specification, a particular array, hereafter referred to as a "history variable" is one which contains a time history dimension; that is, it contains values for the current frame as well as for previous frames. The history variables are the following:

A_ACCELERATION (1:3,0:4)
A_STATUS (1:3,0:3)
AR_ALTITUDE (0:4)
AR_STATUS (0:4)
G_ROTATION (1:3,0:4)
GP_ALTITUDE (0:4)
GP_ATTITUDE (1:3,1:3,0:4)
GP_VELOCITY (1:3,0:4)
K_ALT (0:4)
K_MATRIX (1:3,1:3,0:4)
TDLR_VELOCITY (1:3,0:4)

In each case, the last dimension is the time dimension. The first subscript in a time history dimension is always declared to be zero. The time dimension contains a set of scalars, vectors, or arrays, depending on whether the total number of dimensions is one, two, or three, respectively. Let the term "object" denote a scalar, vector, or array, as appropriate for the particular variable. Each of these variables contains either four or five objects, depending on whether the last dimension is declared to be 0:3 or 0:4 respectively. The variable A_STATUS contains four objects, while each of the other time history variables contains five objects.

Each of the variables listed contains a most recent object and either three or four previous objects. The object with a time subscript of zero is the most recent object; the object with a time subscript of one is the object which is one frame older; the object with a time subscript of two is the object which is two frames older, etc.; the object with the largest time subscript (three or four) is the oldest object.⁷

CONVENTIONS

FORTRAN Convention

This specification was written with the assumption that the implementation would be coded in FORTRAN. If the development language used is something other than FORTRAN, the programmer must investigate the possibility of differences between FORTRAN and the development language chosen.

REQUIREMENTS

Order of Processing

Within each functional unit in Chapter 5, the processing steps are given in a particular order. If the implementation uses the same order as that given in the specification, then correct results should be obtained; however, the programmer is free to use a different order as long as the change in order does not affect the outputs.⁸

Calls to GCS_SIM_RENDEZVOUS

There must be a call to GCS_SIM_RENDEZVOUS prior to the execution of each subframe.⁹

Control Signals

The control signals listed in Table 6.5 in Part III of the Data Requirements Dictionary may be implemented by the programmer in any form desired, or they may be completely ignored and the control of the program may be conducted through other means.

Number Representations

When variables are given in sign-magnitude or other unusual formats, conversion or manipulation may be necessary.¹⁰

Conversion of Units

It is the responsibility of the programmer to be sure that any implied conversion of units is performed.¹¹

Global Data Store Organization

Part II of the Data Requirements Dictionary contains descriptions of four required data stores. Each of these data stores is to be located in a separate, globally accessible data region. The division of the global data stores into four separate regions illustrates the fact these regions have a direct mapping to a specific implementation of GCS on hardware components of an actual lander.(See Figure B.1).

If the implementation is being written in FORTRAN, four labeled common blocks should be declared with the labels GUIDANCE_STATE, EXTERNAL, SENSOR_OUTPUT, and RUN_PARAMETERS,

respectively (See Tables 6.1, 6.2, 6.3, and 6.4). The variables declared in each labeled common block must be in the same order as those in the corresponding table.¹²

Use of Variables That Are Not in the Global Data Stores

A programmer may use variables in addition to the global data store variables; however, if the value of such a variable is dependent upon the values of any global data store variable(s), then the programmer should only use the value of such a variable in the same subframe of the same frame in which it was calculated.

Use of Tables

Some tables have the heading "CURRENT STATE" and "ACTIONS". If the actual state of the variables appears under the "CURRENT STATE" section in the table, then the actions listed in the same line are to be performed. If the actual current state is not represented in any line under the "CURRENT STATE" section of the table, then no action is to be taken.

Rotation of History Variables

In Chapter 5, in certain functional units, an instruction is given to "rotate" specific variables. Table 1.1 illustrates what is meant by rotation. The table is given for a variable with a time dimension of 0:4. For a variable with a time dimension of 0:3, the last line of the table should be ignored. Note that after the variable has been rotated, the new or current object is calculated and placed into the zeroth time history position.¹³

Table 1.1: ROTATION OF VARIABLES¹⁴

TIME HISTORY SUBSCRIPT	Values BEFORE ROTATION	VALUES AFTER ROTATION	VALUE AFTER CALCULATIONS FOR CURRENT FRAME
0	O_{n-1}	X	O_n
1	O_{n-2}	O_{n-1}	O_{n-1}
2	O_{n-3}	O_{n-2}	O_{n-2}
3	O_{n-4}	O_{n-3}	O_{n-3}
4	O_{n-5}	O_{n-4}	O_{n-4}

Note: O_i denotes object that was calculated in frame i

n = current frame number

X = denotes that any value is acceptable

EXCEPTION HANDLING¹⁵

During the execution of a computer program, exception conditions may sometimes occur. The implementation should anticipate or detect certain types of exception conditions and take specific actions. The relevant exception conditions and the actions to be taken are listed below.

Exception Conditions

DIVIDE BY ZERO

A division is performed, but the divisor is equal to zero.

NEGATIVE SQUARE ROOT

A square root is taken, but the argument for the square root is negative.

UPPER OR LOWER LIMIT EXCEEDED

The current value for a data element in the `GUIDANCE_STATE` or `SENSOR_OUTPUT` data store exceeds its upper or lower limit as specified in the range section in the **Data Requirements Dictionary Part I**. The data elements in the `RUN_PARAMETERS` and `EXTERNAL` data stores need not be checked for limit exceeded. In addition, it is not necessary for the functional unit CP to check any data elements for limit exceeded.

Action to be Taken for Each Specified Exception Condition

Write the appropriate output as specified below to the FORTRAN Logical Unit Number 6 and then continue. In the case of `UPPER/LOWER LIMIT EXCEEDED`, do not modify the data element. Note that to "continue" implies that the divide will be executed, or the square root will be taken, or the data element with exceeded limit will be used.

Output to be Generated for Each Exception Condition

The first line of the exception message should appear as follows:

```
" %EXCEPTIONAL-CONDITION-GCS-"<insert specific condition here>
```

where the specific condition is one of the following:

```
"DIVIDE_BY_ZERO"
```

```
"NEGATIVE_SQUARE_ROOT"
```

```
"LOWER_LIMIT_EXCEEDED"
```

```
"UPPER_LIMIT_EXCEEDED"
```

The second line of the exception message should contain the name of the functional unit where the exception condition occurred (i.e. AECLP, ASP, etc.), the name of the actual subroutine where the exception condition occurred, and the current value of the frame counter. Implementations that are coded in FORTRAN should use the following FORTRAN format statement:

```
FORMAT (x, a6, x, a32, x, i4)
```

A third line of the exception message containing information that is specific to the individual error type may be required as specified below.

Divide By Zero

No additional output necessary.

Negative Square Root

Display the value of the argument to the square root operation.

Use FORTRAN format statement `FORMAT (x, e23.14)`.

Lower Limit Exceeded

Display the name of the data element in question and the value of the data element.

Use FORTRAN format statement `FORMAT (x, a32, e23.14)` for type real elements, and use `FORMAT (x, a32, i12)` for integer or logical data elements.

Upper Limit Exceeded

Display the name of the data element in question and the value of the data element.

Use FORTRAN format statement `FORMAT (x, a32, e23.14)` for type real elements, and use `FORMAT (x, a32, i12)` for integer or logical data elements.

2. LEVELS 0 AND 1 SPECIFICATION

LEVEL 0 SPECIFICATION

The GCS will provide an interface between the sensors (rate of descent, attitude, etc.) and the engines (roll and axial). The purpose of the GCS is to keep the vehicle descending along the predetermined velocity-altitude contour which has been chosen to conserve enough fuel to effect a safe attitude and touch down.

The GCS effects this control by:

- processing the following sensor information:
 - acceleration data from the three accelerometers -- one for each vehicle axis,
 - range rate data from four splayed doppler radar beams,
 - altitude data from one altimeter radar,
 - temperature data from a solid-state temperature sensor and a thermocouple pair temperature sensor,
 - rates of rotation from three strapped-down gyroscopes -- one for each vehicle axis, and
 - sensing of touch down by the touch down sensor.
- determining the appropriate commands for the axial and roll engines and the chute release mechanism and issuing them to keep the vehicle on a predetermined velocity-altitude contour.

The GCS also transmits telemetry data and **synchronizes through a rendezvous routine (GCS_SIM_RENDEZVOUS)** with GCS_SIM [10], the simulator and controller.

Note that implementations of the GCS developed from this specification may be executed singly or in parallel. Consequently, only specific system services can be used in an implementation. In particular, a rendezvous routine will be provided and should be invoked, as specified in the implementation notes in Appendix B. In addition, FORTRAN Intrinsic Functions may be used. Other system services and library routines are explicitly excluded from use by the programmer.

Figures 2.2 through 2.5, 3.1, 3.2, and 4.1 through 4.4, and Tables 2.1, 3.1, 4.1, and 4.2 follow Hatley's extension to Structured Analysis (see Appendix A), with the following exceptions and assumptions.

Exceptions:

1. Any data store may appear at more than one level because the processes specified do not communicate directly but only through data stores.
2. Any unlabeled flow between a process and a data store may not necessarily carry all the information in the data store (the actual flow content is defined by the process specification and the Data Requirements Dictionary Part II).

Assumptions:

1. The initial value for control signals is assumed to be "FALSE".
2. In a process activation table (PAT), an empty process cell indicates the process is deactivated.
3. In a PAT, an empty output cell indicates the control signal value remains unchanged.
4. In a PAT, output control signals receive values before any processes are activated and therefore may delay the activation of processes by deactivating their parent process.

An example of assumption 4 is Table 3.1 where setting RENDEZVOUS to "TRUE" delays the activation of the processes of which RUN_GCS is composed until GCS_SIM sets RENDEZVOUS to "FALSE".

Figure 2.1: STRUCTURE OF THE GCS SPECIFICATION

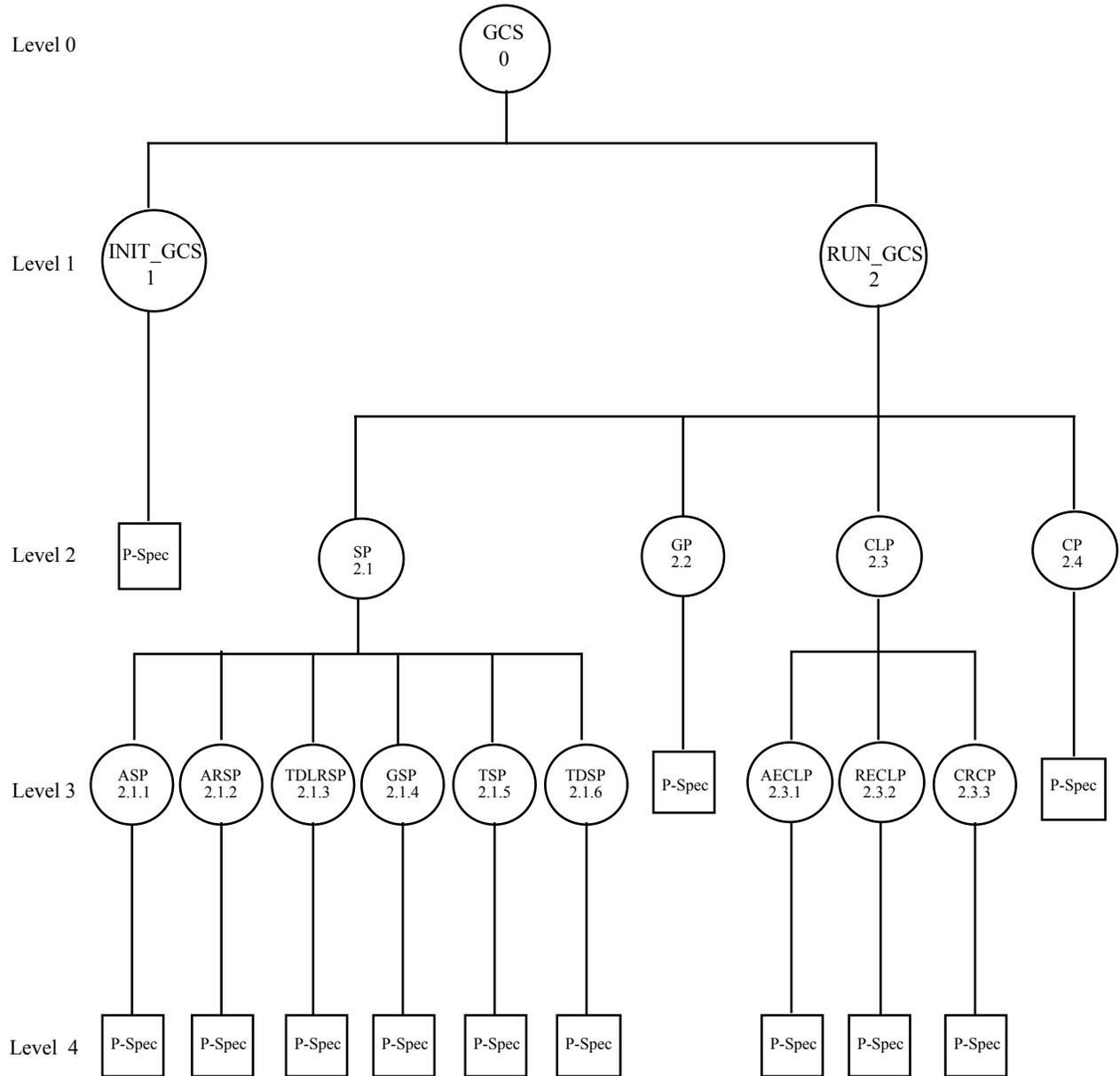


Figure 2.2: DATA CONTEXT DIAGRAM: LANDER

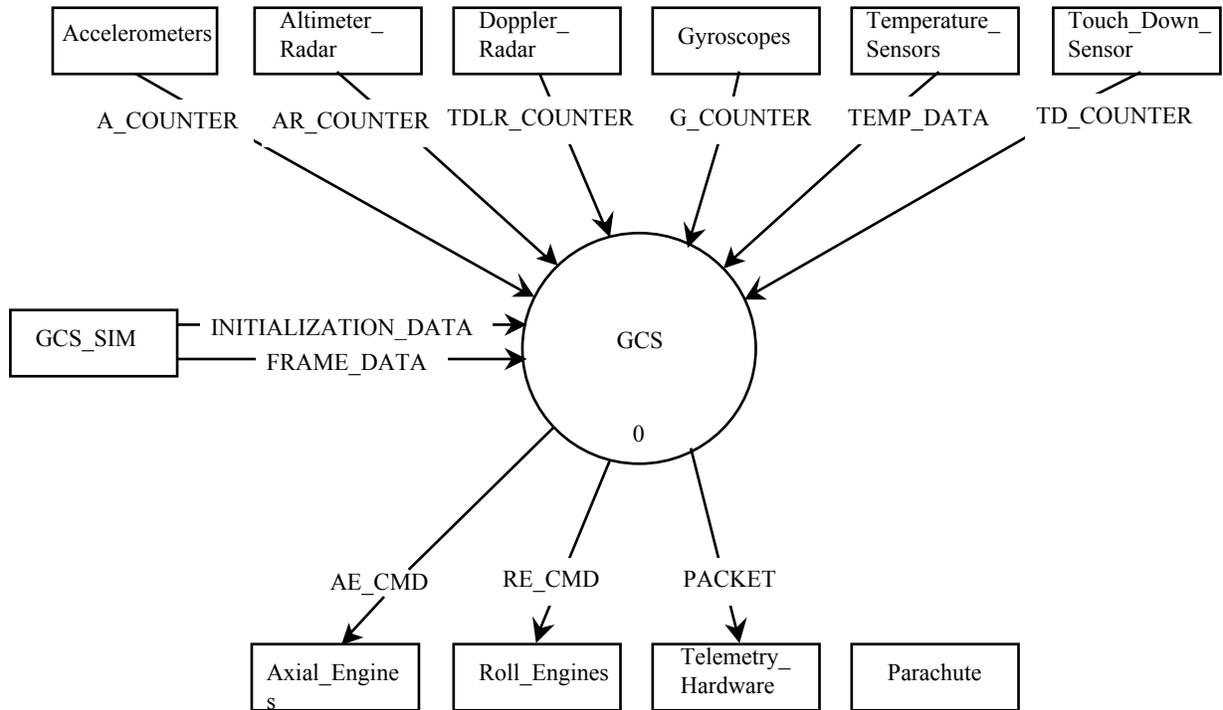
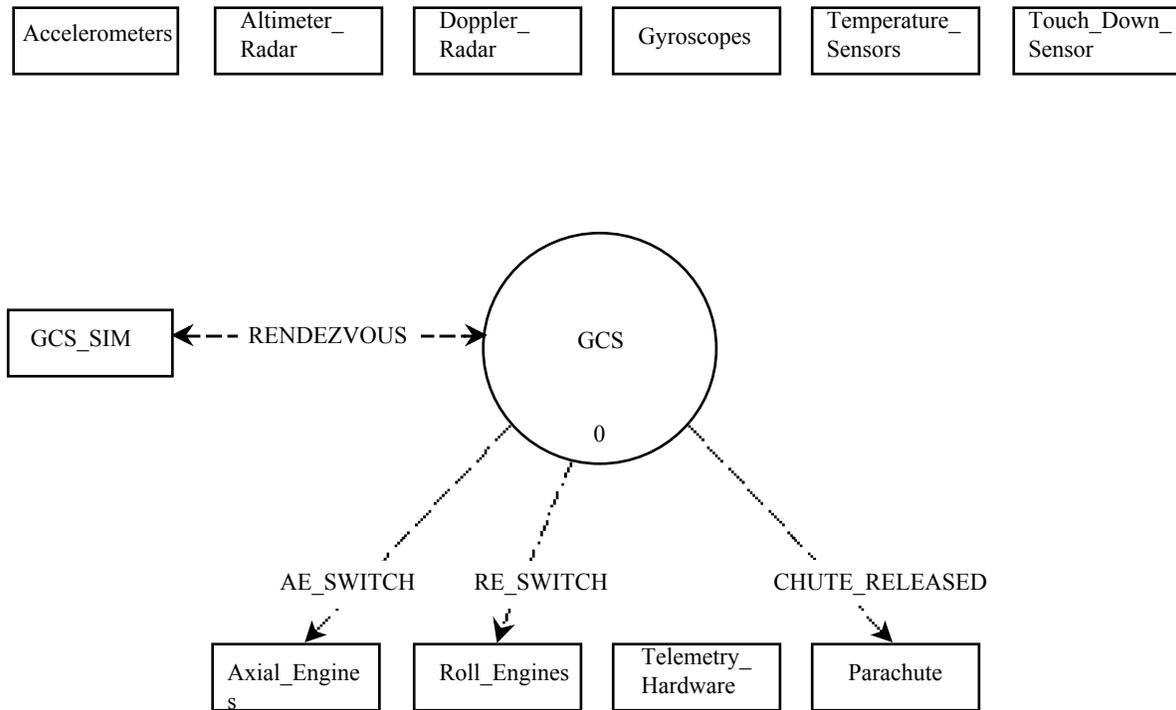


Figure 2.3: CONTROL CONTEXT DIAGRAM: LANDER



LEVEL 1 SPECIFICATION

Figure 2.4: DATA FLOW DIAGRAM (DFD) 0: GCS

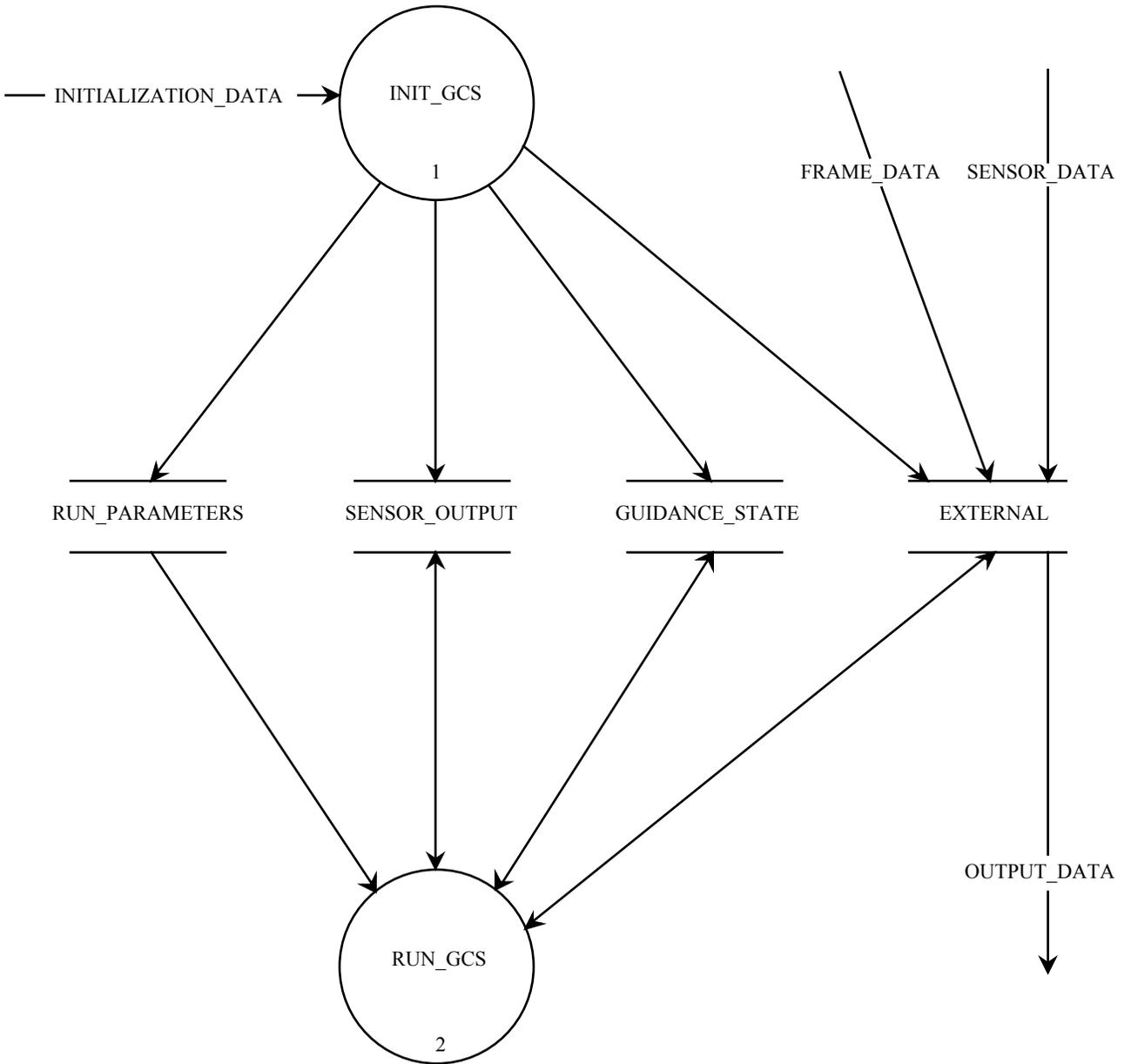
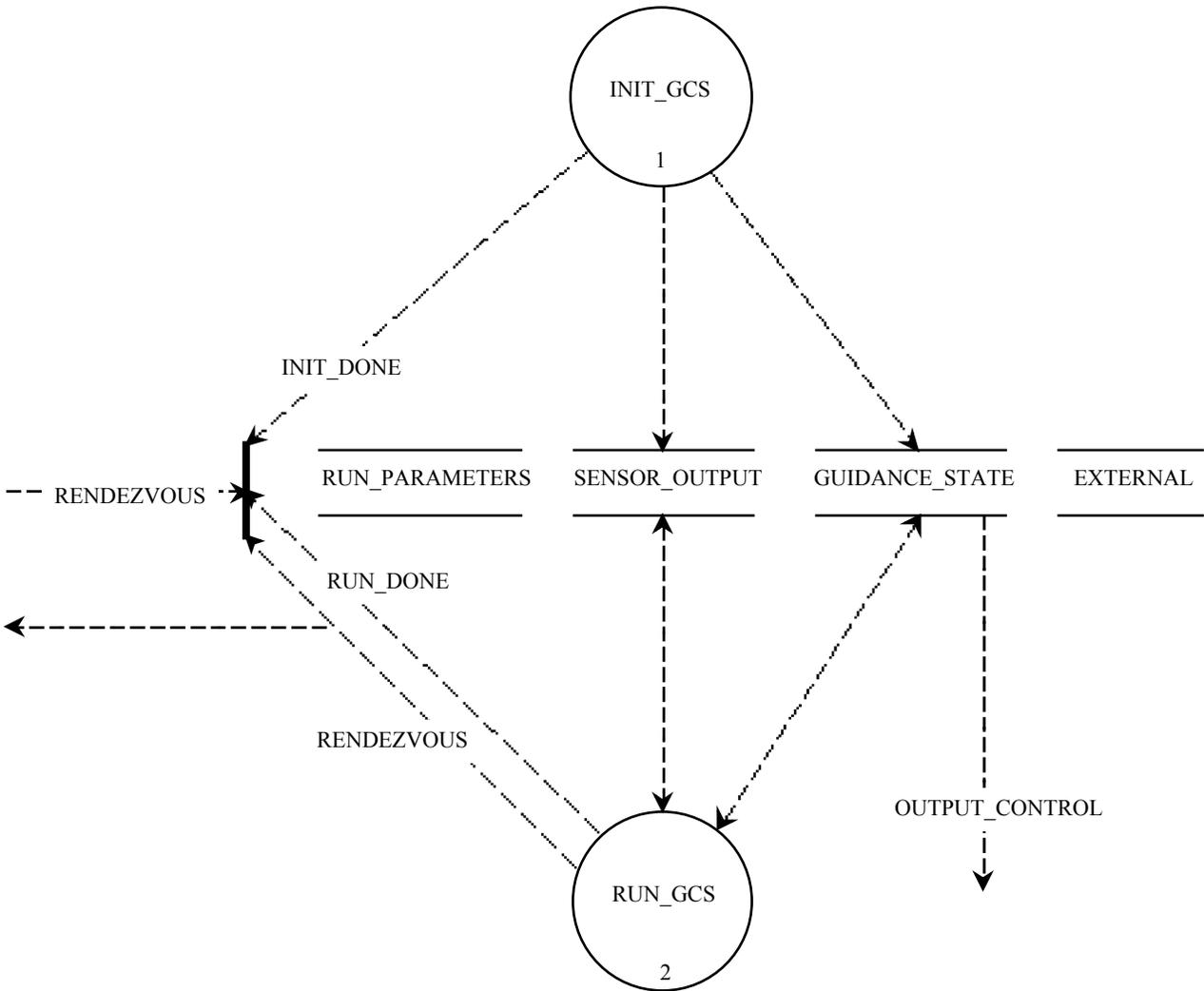


Figure 2.5: CONTROL FLOW DIAGRAM (CFD) 0: GCS



RENDEZVOUS is only set to "TRUE" by RUN_GCS and it is only set to "FALSE" by GCS_SIM.

Table 2.1: CONTROL SPECIFICATION (C-SPEC) 0: GCS

	"INIT_GCS"	"RUN_GCS"
~RENDEZVOUS & ~RUN_DONE		1
RENDEZVOUS & ~INIT_DONE & ~RUN_DONE	1	
(RENDEZVOUS & INIT_DONE) RUN_DONE		

3. LEVEL 2 SPECIFICATION

LEVEL 2 SPECIFICATION

PROCESS SPECIFICATION (P-Spec) 1: INIT_GCS¹⁶

PURPOSE INIT_GCS initializes the guidance and control software.

INPUT

INITIALIZATION_DATA

OUTPUT

INITIALIZATION_DATA

PROCESS INIT_GCS is actually a part of GCS_SIM_RENDEZVOUS, which will be supplied to the programmer; thus the functions performed by INIT_GCS are listed here for information only, but are not the responsibility of the programmer. There should be a call to GCS_SIM_RENDEZVOUS, prior to executing each subframe. **The first call to GCS_SIM_RENDEZVOUS will cause INIT_GCS to automatically be executed.** INIT_GCS will initialize all variables in the group flow INITIALIZATION_DATA, which is defined in Table 6.7 in the Data Requirements Dictionary Part III. Since the variables FRAME_COUNTER and SUBFRAME_COUNTER are part of INITIALIZATION_DATA, they will be initialized at this time. FRAME_COUNTER will be initialized to a value representing the next frame to be executed, while SUBFRAME_COUNTER will always be initialized to the value one, which implies that the first subframe of the first frame to be executed will always be the sensor processing subframe. Although a terminal descent trajectory begins with FRAME_COUNTER initialized to the value one, the option exists for starting execution at some point other than at the beginning of the trajectory, i.e., FRAME_COUNTER may be initialized to a value greater than one.

Figure 3.1: DFD 2: RUN_GCS

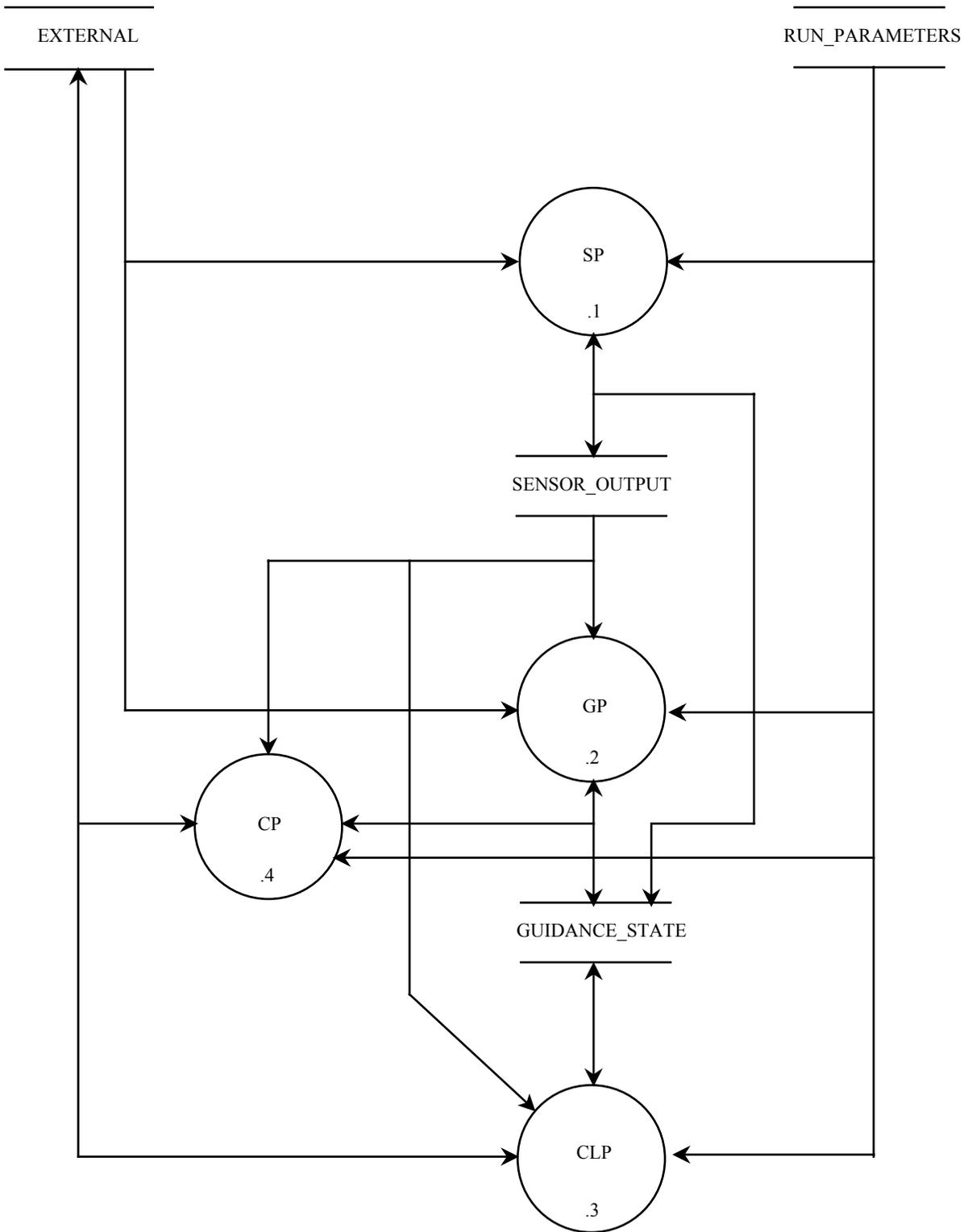


Table 3.1: C-Spec 2: RUN_GCS

	"SP"	"GP"	"CLP"	"CP"	SP_DONE	GP_DONE	CLP_DONE	CP_DONE	RENDEZVOUS	RUN_DONE
~SP_DONE & ~GP_DONE & ~CLP_DONE & ~CP_DONE	1			2					"TRUE"	
SP_DONE & CP_DONE		1		2	"FALSE"			"FALSE"	"TRUE"	
GP_DONE & CP_DONE & GP_PHASE ~= 5			1	2		"FALSE"		"FALSE"	"TRUE"	
CLP_DONE & CP_DONE	1			2			"FALSE"	"FALSE"	"TRUE"	
GP_DONE & CP_DONE & GP_PHASE = 5										"TRUE"

4. LEVEL 3 FLOW DIAGRAMS AND C-SPECS

Figure 4.1: DFD 2.1: SP -- Sensor Processing

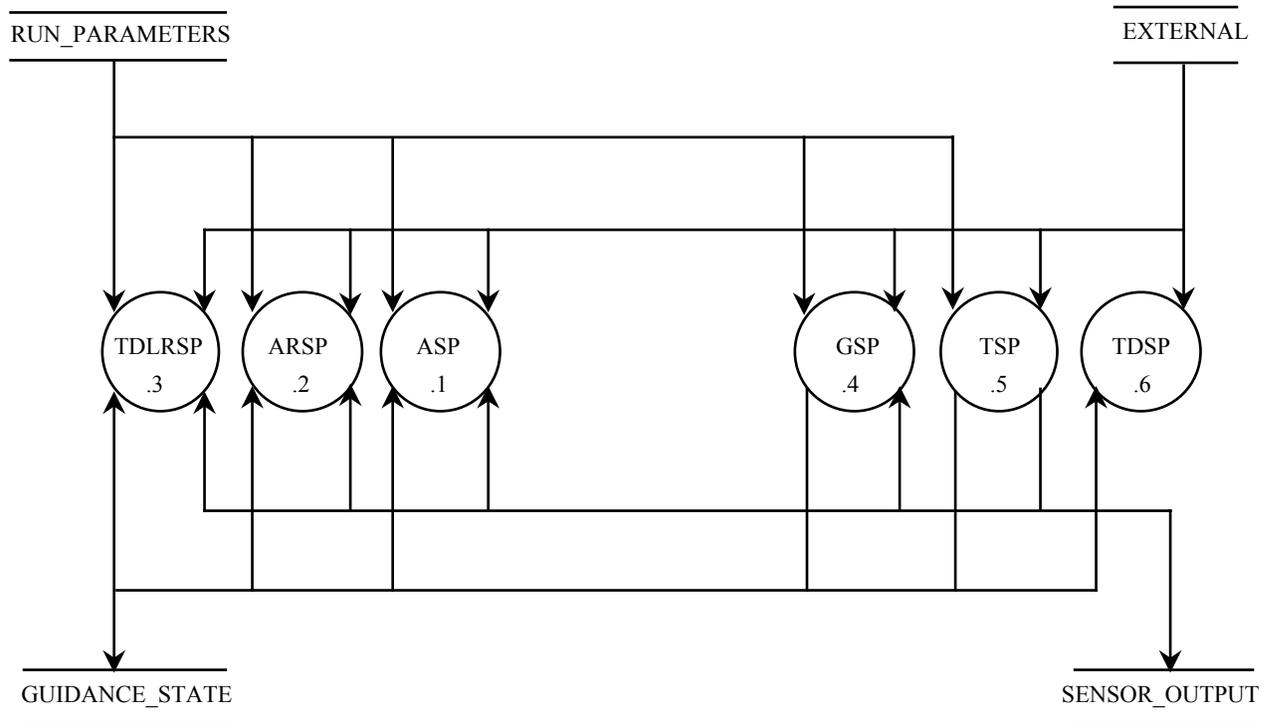


Figure 4.2: CFD 2.1: SP -- Sensor Processing

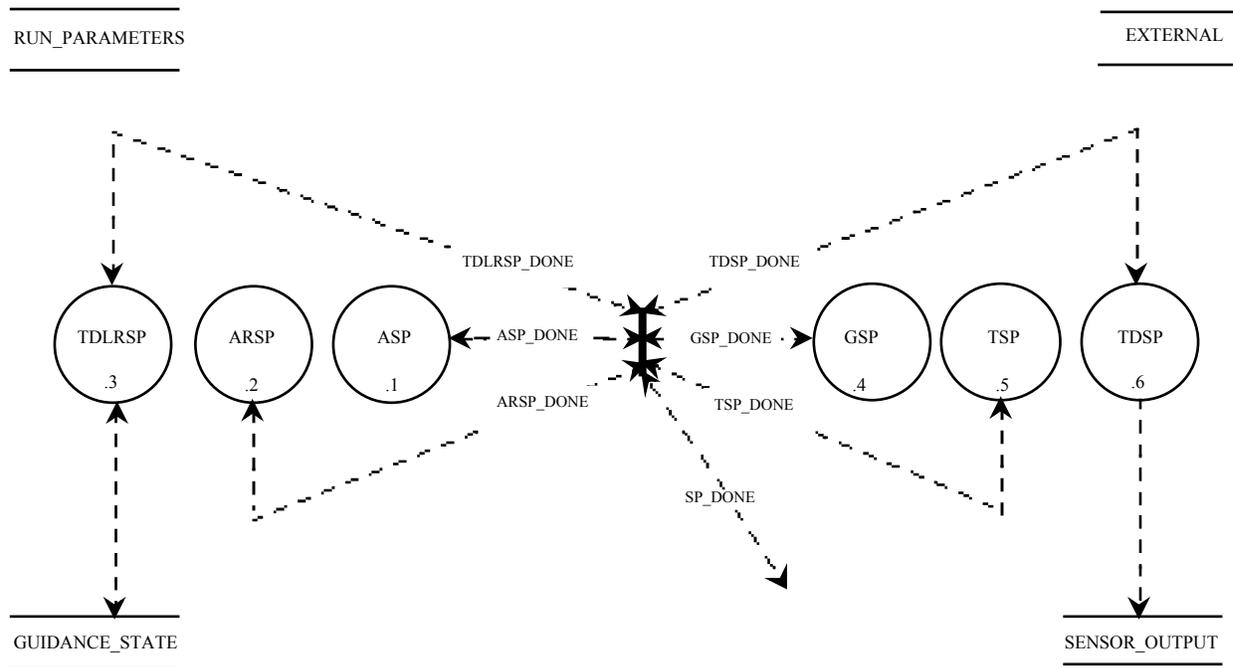


Table 4.1: C-Spec 2.1: SP -- Sensor Processing

	"ASP"	"ARSP"	"TDLRSP"	"GSP"	"TSP"	"TDSP"	ASP_ DONE	ARSP_ DONE	TDLRSP_ DONE	GSP_ DONE	TSP_ DONE	TDSP_ DONE	SP_ DONE
~ASP_DONE & ~ARSP_DONE & ~TDLRSP_DONE & ~GSP_DONE & ~TSP_DONE & ~TDSP_DONE & ~SP_DONE	2	2	2	2	1	2							
ASP_DONE & ARSP_DONE & TDLRSP_DONE & GSP_DONE & TSP_DONE & TDSP_DONE & ~SP_DONE							"FALSE"	"FALSE"	"FALSE"	"FALSE"	"FALSE"	"FALSE"	"TRUE"

Figure 4.3: DFD 2.3: CLP -- Control Law Processing

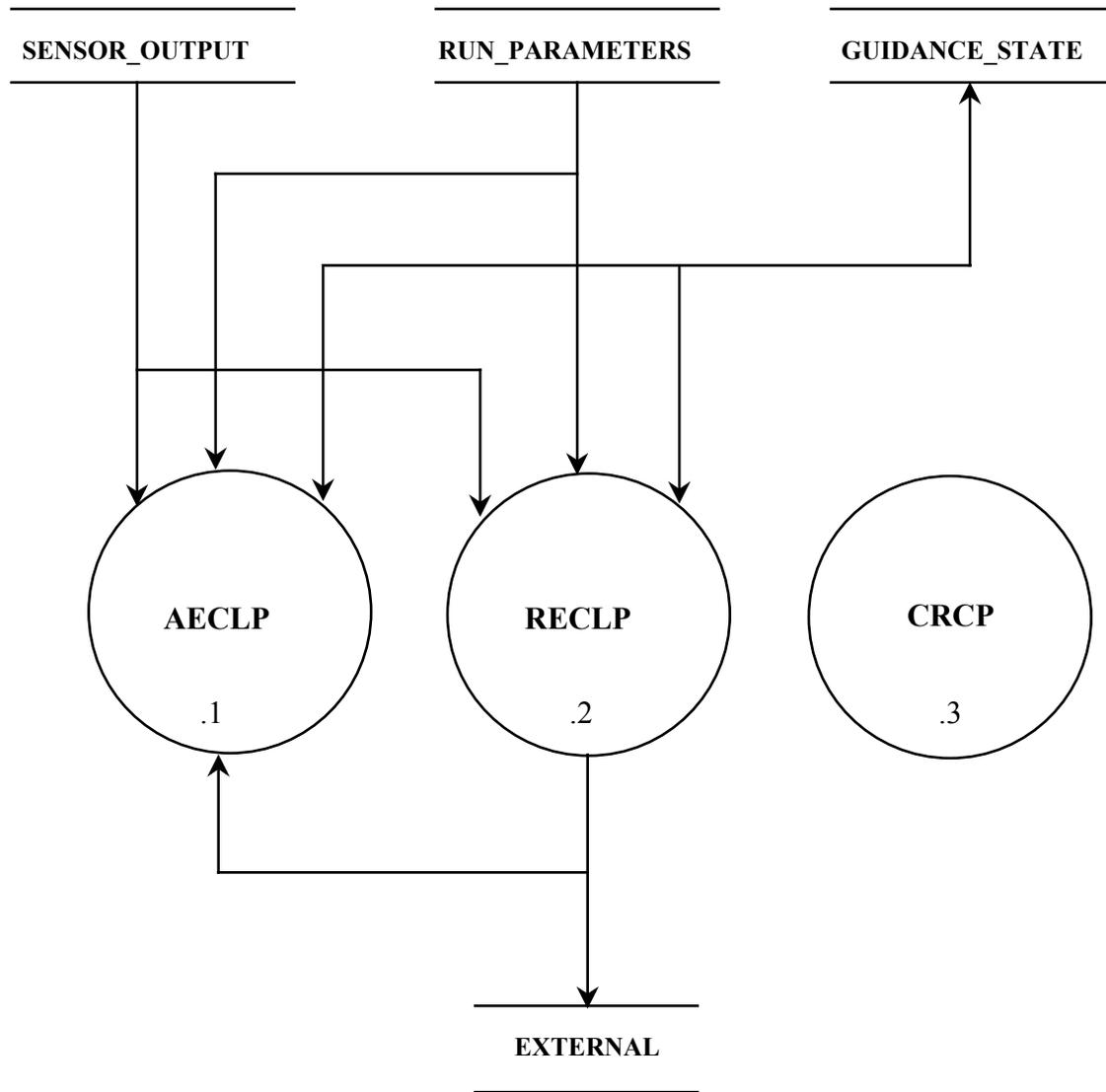


Figure 4.4: CFD 2.3: CLP -- Control Law Processing

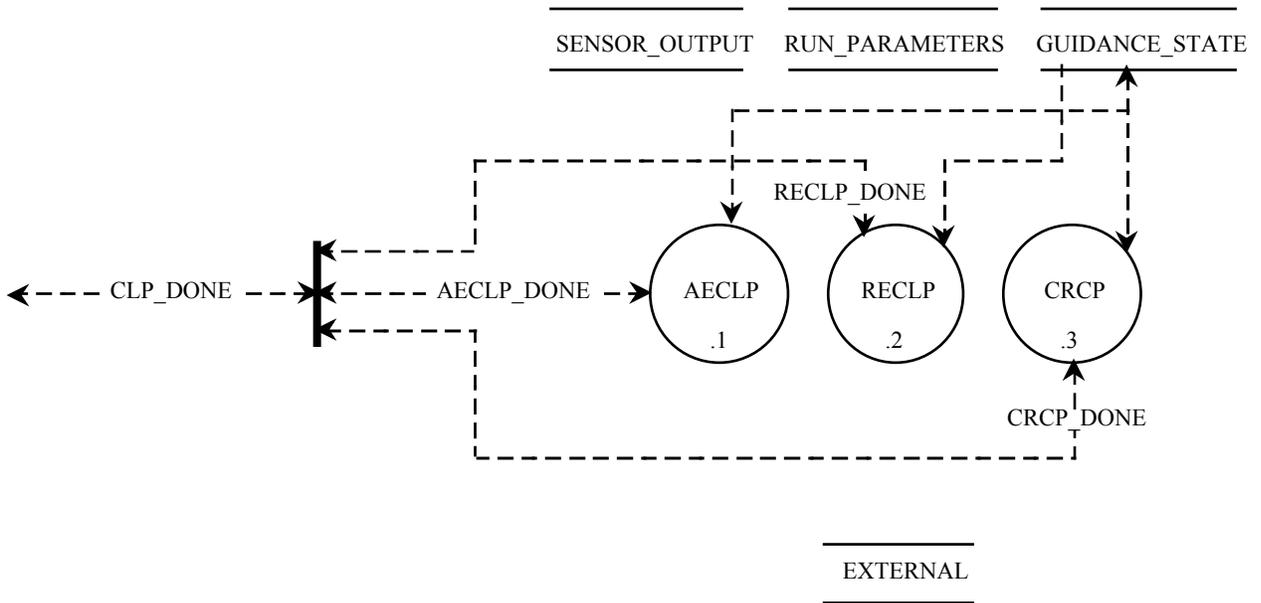


Table 4.2: C-Spec 2.3: CLP -- Control Law Processing

	"AECLP"	"RECLP"	"CRCP"	AECLP_DONE	RECLP_DON E	CRCP_DONE	CLP_DONE
~AECLP_DONE & ~RECLP_DONE & ~CRCP_DONE & ~CLP_DONE	1	1					
AECLP_DONE & ~CRCP_DONE & ~CLP_DONE		1	1				
AECLP_DONE & RECLP_DONE & CRCP_DONE & ~CLP_DONE				"FALSE"	"FALSE"	"FALSE"	"TRUE"

SCHEDULING

Within each frame, the Sensor Processing Subframe is to be executed first, the Guidance Processing Subframe is to be executed second, and the Control Law Processing Subframe is to be the last subframe executed. Table 4.3 lists each functional unit in the GCS according to the subframe in which it should be executed. A number "I" is located along with the functional unit name. This number indicates that the functional unit should be executed every "Ith" frame. Note that all functional units are executed during frame number 1. Also note that execution of the GCS may begin at any frame number and should operate as if it had been running from the beginning of the trajectory (frame number 1). There are minor sequencing constraints to be imposed upon the functional units in each subframe. During the sensor processing subframe, TSP should be executed before any of the other functional units, and CP should be executed last. In the guidance and control subframes, CP should be executed after the other functional units. Lastly, during the control processing subframe, AECLP needs to be executed before CRCP. All functional units not specified here may be executed in any order within their subframes. On the first, and subsequent, calls to GCS_SIM_RENDEZVOUS, FRAME_COUNTER and SUBFRAME_COUNTER will be returned to the implementation containing the correct values for operation. The value in FRAME_COUNTER should be compared to the numbers listed in Table 4.3 to determine if a functional unit should be executed. As an example, TSP has a number of 2, which means that it executes every other frame; while ASP has a number of 1, meaning it executes every frame; and TDSP has a number of 5, so it executes only every fifth frame.

Table 4.3: FUNCTIONAL UNIT SCHEDULING¹⁷

SCHEDULING	
Sensor Processing Subframe (Subframe 1)	"I"
ARSP	1*
ASP	1
CP	1
GSP	1
TDLRSP	1*
TDSP	5
TSP	2
Guidance Processing Subframe (Subframe 2)	"I"
CP	1
GP	1
Control Law Processing Subframe (Subframe 3)	"I"
AECLP	1
CP	1
CRCP	5
RECLP	1

* This functional unit has special scheduling considerations. For details, see the appropriate functional unit description in Chapter 5.

The GCS software must meet all the requirements for a particular frame for any specific value of the variable FRAME_COUNTER. The software must be capable of executing continuously one frame after another until specified termination conditions are met, at which time it must terminate itself according to specified termination procedures.

The termination conditions and procedures are: GCS should check whether to terminate itself in each frame immediately after executing the Guidance Processing functional unit. At that time if the value of the variable GP_PHASE is equal to 5, then GCS should terminate itself gracefully (without any exception conditions). In this case, the implementation should terminate at the end of the present subframe, i.e., it should execute the functional unit Communications Processing and then terminate without calling GCS_SIM_RENDEZVOUS.

5. P-SPECS FOR LEVELS 3 and 4

AECLP -- Axial Engine Control Law Processing (P-Spec 2.3.1)

PURPOSE The AECLP **functional unit** computes the valve settings for each of the three main (axial) engines. Measurements of the vehicle's velocity, acceleration, and roll rates are combined to produce error signals for the pitch, yaw, and thrust of the vehicle. These error signals are then mixed to produce the axial engine valve settings.

INPUT

AE_SWITCH	AE_TEMP
A_ACCELERATION	CHUTE_RELEASED
CL	CONTOUR_CROSSED
DELTA_T	ENGINES_ON_ALTITUDE
FRAME_COUNTER	FRAME_ENGINES_IGNITED
FULL_UP_TIME	GA
GAX	GP1
GP2	GPY
GP_ALTITUDE	GP_ATTITUDE
GP_ROTATION	GP_VELOCITY
GQ	GR
GRAVITY	GV
GVE	GVEI
GVI	GW
GW	OMEGA
PE_INTEGRAL	PE_MAX
PE_MIN	TE_DROP
TE_INIT	TE_INTEGRAL
TE_LIMIT	TE_MAX
TE_MIN	VELOCITY_ERROR
YE_INTEGRAL	YE_MAX
YE_MIN	

OUTPUT

AE_CMD	AE_STATUS
AE_TEMP	INTERNAL_CMD
PE_INTEGRAL	TE_INTEGRAL
TE_LIMIT	YE_INTEGRAL

PROCESS The reader should refer to **Appendix C for notes on integration**. **Note that** once the correct value of AE_CMD has been determined, it will automatically be transmitted to the engines during the next call to the GCS_SIM_RENDEZVOUS routine provided in the GCS_SIM rendezvous package. (See Appendix B. Implementation Notes). Computation of the axial engine valve settings requires the following steps:

- ✓ **PROCESSING WHEN AXIAL ENGINES ARE OFF¹⁹**
 - **IF AE_SWITCH is set to OFF, then perform the following steps:**
 - **Set all elements of AE_CMD to 0**

- **Proceed directly to the step "SET AXIAL ENGINE STATUS TO HEALTHY."**

✓ **PROCESSING WHEN AXIAL ENGINES ARE ON**²⁰

The variable CL is used here as a subscript. Explanations for the variables CL and VELOCITY_ERROR are provided in functional unit 2.6 GP. The variables PE_INTEGRAL, YE_INTEGRAL, and TE_INTEGRAL will be initialized by INIT_GCS.

- If AE_SWITCH is set to ON then perform the following steps:

(Note: p_v , q_v , and r_v are the current elements of GP_ROTATION; \dot{x}_v , \dot{y}_v , and \dot{z}_v are the current elements of GP_VELOCITY; \ddot{x}_v is the current x component of A_ACCELERATION.)

DETERMINE ENGINE TEMPERATURE

- Set AE_TEMP according to Table 5.1

Table 5.1: DETERMINATION OF AXIAL ENGINE TEMPERATURE²¹

CURRENT STATE			ACTION
AE_TEMP	GP_ALTITUDE	(FRAME_COUNTER – FRAME_ENGINES_IGNITED DELTA_T	AE_TEMP
cold	≤ ENGINES_ON_ALTITUDE	< FULL_UP_TIME	warming-up
warming-up	≤ ENGINES_ON_ALTITUDE	≥ FULL_UP_TIME	hot

COMPUTE LIMITING ERRORS FOR PITCH

••
$$PE_INTEGRAL = PE_INTEGRAL + \int_{t_0}^t \frac{\dot{x}_v}{|\dot{x}_v|} dt$$

where t_0 is the beginning of the time step and t is the end of the time step.

..
$$P_e^L = GQ(CL) \cdot q_v + GW(CL) \cdot \left(\frac{\dot{X}_v}{|\dot{X}_v|} \right) + GWI(CL) \cdot PE_INTEGRAL$$

.. If $P_e^L < PE_MIN(CL)$ then set P_e^L to $PE_MIN(CL)$.

.. If $P_e^L > PE_MAX(CL)$ then set P_e^L to $PE_MAX(CL)$.

COMPUTE LIMITING ERROR FOR YAW

••
$$YE_INTEGRAL = YE_INTEGRAL + \int_{t_0}^t \frac{\dot{Y}_v}{|\dot{Y}_v|} dt$$

where t_0 is the beginning of the time step and t is the end of the time step.

••
$$Y_e^L = -GR(CL) \cdot r_v + GV(CL) \cdot \left(\frac{\dot{Y}_v}{|\dot{Y}_v|} \right) + GVI(CL) \cdot YE_INTEGRAL$$

•• If $Y_e^L < YE_MIN(CL)$ then set Y_e^L to $YE_MIN(CL)$.

•• If $Y_e^L > YE_MAX(CL)$ then set Y_e^L to $YE_MAX(CL)$.

COMPUTE LIMITING ERROR FOR THRUST

•• If **CONTOUR_CROSSED** is set to "contour not crossed", then proceed directly to the step "COMPUTE PITCH, YAW, AND THRUST ERRORS."

•• If **CONTOUR_CROSSED** is set to "contour crossed", then perform the following steps:

•••
$$TE_INTEGRAL = TE_INTEGRAL + \int_0^t (VELOCITY_ERROR) dt$$

••• Solve the following equation analytically in order to calculate the value for **TE_LIMIT**:

$$\frac{d}{dt}(TE_LIMIT) + OMEGA \cdot TE_LIMIT = \frac{-GAX \cdot \ddot{X} + GRAVITY \cdot GP_ATTITUDE(1,3,0) + GVE \cdot VELOCITY_ERROR + GVEI(CL) \cdot TE_INTEGRAL}{GA}$$

... If $TE_LIMIT < TE_MIN(CL)$ then set TE_LIMIT to $TE_MIN(CL)$.

... If $TE_LIMIT > TE_MAX(CL)$ then set TE_LIMIT to $TE_MAX(CL)$.

COMPUTE PITCH, YAW, AND THRUST ERRORS

- Compute pitch error (P_e), Yaw Error (Y_e), and Thrust Error (T_e), according to Table 5.2

Table 5.2: DETERMINATION OF ERROR TERMS²³

AE_SWITCH	CHUTE_ RELEASED	CONTOUR_ CROSSED	P_e	Y_e	T_e
1	1	1	P_e^L	Y_e^L	TE_LIMIT
1	1	0	P_e^L	Y_e^L	TE_DROP
1	0	0,1	$GQ(CL) \cdot q_v$	$-GR(CL) \cdot r_v$	TE_INIT

COMPUTE AXIAL ENGINE VALVE SETTINGS

Given pitch, yaw, and thrust errors, (P_e, Y_e, T_e), the valve settings (AE_CMD) for each of the three main engines are calculated as:

$$\text{INTERNAL_CMD} = \begin{pmatrix} GP1 & 0 & 1 \\ GP2 & -GPY & 1 \\ GP2 & GPY & 1 \end{pmatrix} \times \begin{pmatrix} P_e \\ Y_e \\ T_e \end{pmatrix}$$

which will result in each element of the INTERNAL_CMD vector being a real value. This value should be converted into an integer value between 0 and 127 and placed into the appropriate element of the AE_CMD vector. The mapping for the conversion from real to integer values **for each of the three elements** should be as follows:

Table 5.3: DETERMINATION OF AXIAL ENGINE COMMANDS

INTERNAL_CMD	AE_CMD
$I < 0.0$	$A = 0$
$0.0 \leq I \leq 1.0$	$0 \leq A \leq 127$
$1.0 < I$	$A = 127$

Note: "I" represents the appropriate element of the vector INTERNAL_CMD
 "A" represents the appropriate element of the vector AE_CMD

with INTERNAL_CMD between 0 and 1.0 being converted *linearly* to a value of AE_CMD between 0 and 127. **Each value for AE_CMD is to be rounded to the nearest integer, where rounding is defined as follows:**²⁵

Let x represent the real value that is to be rounded

Then, $AE_CMD = \text{the integer part of } (x+0.5)$

- ✓ SET AXIAL ENGINE STATUS TO HEALTHY
 - Set AE_STATUS to healthy.

ARSP -- Altimeter Radar Sensor Processing (P-Spec 2.1.2)

PURPOSE The vehicle has one altimeter radar. The ARSP **functional unit** reads the altimeter counter provided by this radar and converts the data into a measure of distance to the surface.

INPUT

AR_ALTITUDE	AR_COUNTER
AR_FREQUENCY	AR_STATUS
FRAME_COUNTER	K_ALT

OUTPUT

AR_ALTITUDE	AR_STATUS
K_ALT	

PROCESS It is only necessary that this functional unit perform its normal calculations every other frame, namely on the odd-numbered frames; however, one will notice that in the scheduling Table 4.1, it is required that this functional unit execute every frame. The reason for this is that during its normal processing it must rotate history variables. This means that during the frames when it does not need to calculate new outputs, namely the even-numbered frames, it must still rotate its history variables and set its new or current values equal to the previous values, thus creating double entries for each rotated variable. By doubling the entries, consistency of time histories will be maintained at the expense of keeping two copies of each value in these variables, and forcing the functional unit to execute every frame.²⁶

The processing of the altimeter counter data (AR_COUNTER) into the vehicle's altitude above the planet's terrain depends on whether or not an echo is received by the altimeter radar for the current time step. The distance covered by the radio pulses emitted from the altimeter radar is directly proportional to the time between transmission and reception of its echo. A **digital counter** (AR_COUNTER) is started as the radar pulse is transmitted. The counter increments AR_FREQUENCY times per second. **If an echo is received, the lower order fifteen bits of AR_COUNTER contain the pulse count, and the sign bit will contain the value zero. If an echo is not received, AR_COUNTER will contain sixteen one bits.**²⁷

✓ ROTATE VARIABLES

- Rotate AR_ALTITUDE, AR_STATUS, AND K_ALT.

✓ PERFORM ALTERNATE PROCESSING IF THIS IS AN EVEN-NUMBERED FRAME

- If FRAME_COUNTER is an even number, then perform the following:
 - Insure that the current values of AR_ALTITUDE, AR_STATUS, and K_ALT are equal to the previous values of AR_ALTITUDE, AR_STATUS, and K_ALT respectively.
 - Exit from this functional unit.

✓ DETERMINE ALTITUDE²⁸

- If an echo is received, perform the following:
 - Convert the AR_COUNTER value to a distance to be returned in the variable AR_ALTITUDE according to the following equation:

$$AR_ALTITUDE = \frac{AR_COUNTER \cdot 3 \times 10^8 \frac{m}{sec}}{AR_FREQUENCY \cdot 2}$$

- If an echo is not received, compute AR_ALTITUDE as follows:
 - If all four previous values of AR_STATUS are healthy:
 - In order to smooth the estimate of altitude, fit a third-order polynomial to the previous four values of AR_ALTITUDE.
 - Use this polynomial to extrapolate a value for AR_ALTITUDE for the current time step.
 - If any of the previous four values of AR_STATUS is failed:
 - Set the current value of AR_ALTITUDE equal to the previous value of AR_ALTITUDE.

✓ SET ALTIMETER RADAR STATUS

- Set the current values for AR_STATUS and K_ALT according to TABLE 5.4.

Table 5.4: DETERMINATION OF ALTITUDE STATUS²⁹

CURRENT STATE		ACTIONS TO BE TAKEN	
ECHO RETURNED?	All 4 previous AR_STATUS values healthy?	AR_STATUS	K_ALT
yes	d	healthy	1
no	yes	failed	1
no	no	failed	0

Note: "d" = don't care condition

ASP -- Accelerometer Sensor Processing (P-Spec 2.1.1)

PURPOSE Three accelerometers, located at the vehicle's center of gravity, are slightly misaligned along the vehicle's \vec{x}_v , \vec{y}_v , and \vec{z}_v axes. Each accelerometer produces a 16-bit binary value (A_COUNTER), represented as the magnitude portion of a sign magnitude number which is a linear function of the acceleration along its axis. The sign of the counter will always be positive, but the offset given in A_BIAS will be negative or zero, so if the magnitude of the product of A_COUNTER and A_GAIN is smaller than that of A_BIAS, the **measured** acceleration is negative. The Acceleration Sensor Processing (ASP) **functional unit** provides measures of the vehicle accelerations through the conversion and digital filtering of this raw accelerometer data.

INPUT

A_ACCELERATION	A_BIAS
A_COUNTER	A_GAIN_0
A_SCALE	A_STATUS
ALPHA_MATRIX	ATMOSPHERIC_TEMP
G1	G2

OUTPUT

A_ACCELERATION	A_STATUS
----------------	----------

PROCESS The processing of the accelerometer data (A_COUNTER) into vehicle accelerations (A_ACCELERATION) requires the following steps:

- ✓ **ROTATE VARIABLES³⁰**
 - **Rotate A_ACCELERATION and A_STATUS.**

- ✓ **ADJUST GAIN FOR TEMPERATURE**

The standard gain (A_GAIN_0) must be adjusted for the effects of temperature prior to the conversion of the raw accelerometer values. The adjusted gain is a quadratic function of the ambient temperature (ATMOSPHERIC_TEMP) and the standard gain.

- **Adjust the gain for temperature as follows:**

$$A_GAIN(i) = A_GAIN_0(i) + (G1 \cdot ATMOSPHERIC_TEMP) + (G2 \cdot ATMOSPHERIC_TEMP^2)$$

where i ranges from 1 to 3 and represents the three directions x, y, and z, and where A_GAIN_0 is the standard gain.

✓ REMOVE CHARACTERISTIC BIAS

Each accelerometer has a characteristic DC bias (A_BIAS) which must be removed from the signal prior to conversion. The acceleration is a linear function of its A_COUNTER value where the gain specifies the slope and the offset (A_BIAS) specifies the intercept.

- **Remove the bias as follows:**

$$A_ACCELERATION_M(i) = A_BIAS(i) + A_GAIN(i) * A_COUNTER(i)$$

where i ranges from 1 to 3 and represents the three directions x, y, and z.

✓ CORRECT FOR MISALIGNMENT

Each accelerometer is slightly misaligned from the true vehicle axes. The multiplier matrix (**ALPHA_MATRIX**) which is shown below, is based on small angle approximations and corrects for this misalignment. It is used for transforming the measured acceleration data into the true vehicle accelerations.

$$ALPHA_MATRIX = \begin{pmatrix} 1 & -\alpha_{xz} & \alpha_{xy} \\ \alpha_{yz} & 1 & -\alpha_{yx} \\ -\alpha_{zy} & \alpha_{zx} & 1 \end{pmatrix}$$

α_{xy} defines the angle of rotation about the vehicle's \vec{y}_v axis between the \vec{x}_v axis and the misaligned \vec{x}_v axis. The other misalignment angles are defined similarly, based upon a right-handed coordinate system.

- **Compute preliminary current value of A_ACCELERATION as follows:**

$$A_ACCELERATION = ALPHA_MATRIX \times A_ACCELERATION_M$$

✓ DETERMINE ACCELERATIONS AND ACCELEROMETER STATUS³¹

The variable A_STATUS is a four-element array in each of the three physical dimensions, and contains the present and previous three values of status for each accelerometer. The variable A_ACCELERATION is a five-element array in each of the three dimensions (x, y, and z). A_ACCELERATION contains the present and previous four values of acceleration.

- **The following steps are described for the x axis but should be performed for each axis:**
 - If one or more of the previous three values of A_STATUS is unhealthy, **leave the current value of A_ACCELERATION unchanged**, set the current value of A_STATUS to healthy **and do no further processing for this axis.**
 - If **all three** of the previous values of A_STATUS are healthy, check for extreme values and set A_STATUS and A_ACCELERATION according to the **method described** below. The

accelerometer processing includes filtering of the calculated accelerations along each axis (i.e. filtering of $(\ddot{X}_v, \ddot{Y}_v, \ddot{Z}_v)_t$), and ignoring or eliminating calculated accelerations which are out of range. To effect this filtering, the means and standard deviations for each component of acceleration are to be computed using the calculated accelerations from the previous three time steps. That is, for the current time step t and the measurement of acceleration along the x axis:

••• **Calculate**

$$\hat{\mu} = \sum_{i=t-3}^{t-1} \frac{\ddot{X}_v(i)}{3}$$

which is the current sample mean

••• **Calculate**

$$\hat{\sigma} = \sqrt{\sum_{i=t-3}^{t-1} \frac{(\ddot{X}_v(i))^2}{3} - \hat{\mu}^2}$$

which is the current sample standard deviation.

••• If $|\hat{\mu} - \ddot{X}_v(t)| > A_SCALE \cdot \hat{\sigma}$

set $\ddot{X}_v(t)$ to $\hat{\mu}$

set A_STATUS to unhealthy

where $\ddot{X}_v(t)$ is the acceleration along the x axis for the current time step. Similar equations hold for eliminating outliers in the measures of acceleration along the y and z axes.

otherwise

set A_STATUS to healthy

In summary, if the **calculated acceleration** for the current time step for any component differs from the mean by more than A_SCALE times the standard deviation, then that component **of acceleration** should be replaced by its current mean and A_STATUS should be set to unhealthy.

If the calculated acceleration for any component is within the specified range of the mean, then the preliminary value of $A_ACCELERATION$ should remain unchanged and A_STATUS should be set to healthy.

CP -- Communications Processing (P-Spec 2.4)

PURPOSE Data from the vehicle sensors and guidance processor is relayed back to the orbiting platform for later analysis. The CP **functional unit** converts the sensed data into a data packet appropriate for radio transmission.

INPUT

AE_CMD	AE_STATUS
AE_TEMP	AR_ALTITUDE
AR_STATUS	ATMOSPHERIC_TEMP
A_ACCELERATION	A_STATUS
CHUTE_RELEASED	COMM_SYNC_PATTERN
CONTOUR_CROSSED	FRAME_COUNTER
GP_ALTITUDE	GP_ATTITUDE
GP_PHASE	GP_ROTATION
GP_VELOCITY	G_ROTATION
G_STATUS	K_ALT
K_MATRIX	PE_INTEGRAL
RE_CMD	RE_STATUS
SUBFRAME_COUNTER	TDLR_STATE
TDLR_STATUS	TDLR_VELOCITY
TDS_STATUS	TD_SENSED
TE_INTEGRAL	TS_STATUS
VELOCITY_ERROR	YE_INTEGRAL

OUTPUT

C_STATUS	PACKET
----------	--------

PROCESS The data packet (PACKET) prepared for transmission is organized to sequentially contain a synchronization pattern, a sequence number, new sample mask, the data itself, and the checksum information. The data packet created will automatically be transmitted during the next call to GCS_SIM_RENDEZVOUS.

- ✓ SET COMMUNICATOR STATUS TO HEALTHY
 - **Set C_STATUS to healthy.**

The construction of the packet requires the following steps:

- ✓ CONSTRUCT PACKET:
 - GET SYNCHRONIZATION PATTERN

The synchronization pattern is provided in the variable COMM_SYNC_PATTERN. It is a 16-bit pattern dictated by the design of the receiving communications equipment.

- DETERMINE SEQUENCE NUMBER

The sequence number identifies the packet of data that is being sent. It is a byte value in the range 0..255. The sequence number will be 0 during the first subframe of frame number 1. Sequence numbers increase by one every subframe, except that the values repeat after the 256th packet. The sequence number can be calculated based on the values of the variables FRAME_COUNTER and SUBFRAME_COUNTER.

- PREPARE SAMPLE MASK

The sample mask is a boolean vector where "ones" represent variables that have been sampled since the previous transmission. Any variables listed in Table 5.5 that may have changed during the present subframe should be marked in the mask and transmitted. The output variables from the functional units ARSP and TDLRSP, however, should not be transmitted when the variable FRAME_COUNTER is an even number. Values that have been rotated into subsequent elements of an array are not considered "new" and thus do not have to be transmitted. This eliminates the need to maintain previous values on all variables and also eliminates making comparisons to determine which variables should be sent. Each bit position in the mask represents a particular variable listed in Table 5.5. The leftmost bit of the mask corresponds to AE_CMD, and moving across the mask from left to right, the next mask bit corresponds to the next variable in Table 5.5 (in row order).

- PREPARE DATA SECTION

The data section of the packet contains the sixteen bit values for the elements of the variables in Table 5.5 that may have new samples available. Values that have been rotated into subsequent elements of an array are not considered "new" and thus do not have to be transmitted. Once it has been determined which variables should be transmitted for this particular subframe, those variables should be packed into the data section. Although the length of the variable PACKET is fixed, the number of bytes of PACKET which contain actual variables to be transmitted will vary depending on the values of FRAME_COUNTER and SUBFRAME_COUNTER. The variables to be transmitted should be concatenated so that there are no unused bytes between the data to be transmitted. There may however be unused bytes following the checksum. The data are concatenated in the order given by the sample mask, starting with the most significant bit (i.e. left most bit). Variables should be packed to the nearest byte boundary; thus, a single element of PACKET could contain a logical*1 and the first byte of the variable that follows it. Arrays should be sent with the first index changing most rapidly. It should be noted that some arrays have terms that are constant (e.g. the off-diagonal terms of K_MATRIX and the diagonal terms of GP_ROTATION) and since these terms can never have "new" values, they should not be

transmitted. The values in Table 5.5 should be sent in row order, starting at the top of the table. The first value in alphabetical order goes next to the mask in the packet.³²

- CALCULATE CHECKSUM

The data checksum is calculated on the entire packet (excluding the checksum) using the standard CRC-16 polynomial as defined in [11]. The calculation of the checksum should begin with the COMM_SYNC_PATTERN portion of PACKET, and conclude with the last variable to be sent during the current subframe. Any unused parts of PACKET should be ignored for the calculation of the checksum. The checksum should be placed in the two bytes immediately following the last byte of actual data to be transmitted for this subframe.

Table 5.5: PACKET VARIABLES

AE_CMD	AE_STATUS	AE_TEMP
AR_ALTITUDE	AR_STATUS	ATMOSPHERIC_TEMP
A_ACCELERATION	A_STATUS	CHUTE_RELEASED
CONTOUR_CROSSED	C_STATUS	GP_ALTITUDE
GP_ATTITUDE	GP_PHASE	GP_ROTATION
GP_VELOCITY	G_ROTATION	G_STATUS
K_ALT	K_MATRIX	PE_INTEGRAL
RE_CMD	RE_STATUS	TDLR_STATE
TDLR_STATUS	TDLR_VELOCITY	TDS_STATUS
TD_SENSED	TE_INTEGRAL	TS_STATUS
VELOCITY_ERROR	YE_INTEGRAL	

Note: when read by rows, this table represents the alphabetical listing of variables that are to appear in the data section of the packet.

Table 5.6: SAMPLE MASK

INFORMATION SENT	A	B	C	...	Z
EXAMPLE MASK	1	1	0	...	1

Note: this table gives information only on the order of the packet. The packet should be packed to a byte-boundary limit into integer*2 elements.

Table 5.7: EXAMPLE OF PACKET

COMM_SYNC_PATTERN .
SEQUENCE NUMBER
SAMPLE MASK .
DATA SECTION containing the variables that may have changed since last packet .
CHECKSUM .

Note: this table is one byte wide, but any section containing three vertical dots represents one that may be more than one byte long (e.g. DATA SECTION). Also note that the variables inserted into PACKET are inserted in the VAX standard byte order.

CRCP -- Chute Release Control Processing (P-Spec 2.3.3)

PURPOSE The CRCP **functional unit** implements the release of the parachute which is attached **prior to** the beginning of the terminal descent phase.

INPUT

AE_TEMP	CHUTE_RELEASED
---------	----------------

OUTPUT

CHUTE_RELEASED

PROCESS If the chute has been released, leave CHUTE_RELEASED **unchanged** and this signal will be automatically transmitted to the chute release mechanism during the next call to **GCS_SIM_RENDEZVOUS**. If the chute has not been released, the engine temperature will determine whether or not to release the chute. **If the chute has not been released and** the engines are hot (i.e. AE_TEMP is HOT), then release the chute by setting CHUTE_RELEASED to "**chute released.**"

GP -- Guidance Processing (P-Spec 2.2)

PURPOSE GP uses the information available from ASP, ARSP, CRCP, GSP, TDLRSP, and TDSP and the results of its previous computations to control the vehicle's state during terminal descent.

INPUT

A_ACCELERATION	AE_SWITCH
AE_TEMP	AR_ALTITUDE
CHUTE_RELEASED	CL ³³
CONTOUR_ALTITUDE	CONTOUR_CROSSED
CONTOUR_VELOCITY	DELTA_T
DROP_HEIGHT	DROP_SPEED ³⁴
ENGINES_ON_ALTITUDE	FRAME_COUNTER
GP_ALTITUDE	GP_ATTITUDE
GP_PHASE	GP_VELOCITY
GRAVITY	G_ROTATION
K_ALT	K_MATRIX
MAX_NORMAL_VELOCITY ³⁵	RE_SWITCH
TD_SENSED	TDLR_VELOCITY
TDS_STATUS	

OUTPUT

AE_SWITCH	CL ³⁶
CONTOUR_CROSSED	FRAME_ENGINES_IGNITED
GP_ALTITUDE	GP_ATTITUDE
GP_PHASE	GP_ROTATION
GP_VELOCITY	RE_SWITCH
TE_INTEGRAL ³⁷	VELOCITY_ERROR

ARRAYS The variables GP_ATTITUDE, GP_ALTITUDE, and GP_VELOCITY are five element arrays in each of their history dimensions and contain enough previous values to provide the required history for integration in updating the vehicle and guidance states.

PROCESS The Guidance Processor computes the velocity, altitude, and attitude to be used in controlling the engines.

✓ **ROTATE VARIABLES**³⁸

- **Rotate GP_ATTITUDE, GP_ALTITUDE, and GP_VELOCITY.**

✓ **SET UP THE GP_ROTATION MATRIX**

G_ROTATION contains three values: p, q, and r, **in that order**. These values must be placed into a

3 x 3 matrix (**GP_ROTATION**) in the correct positions for later calculations. Note that GP_ROTATION does not include any time histories; thus it may be convenient to use a temporary variable during calculation to hold the time histories of GP_ROTATION or to use elements directly from G_ROTATION; however, GP_ROTATION does describe the correct matrix orientation for operations and upon exiting from GP should contain the correct values for the present time step.

- **Place the values from G_ROTATION into GP_ROTATION as shown:**

$$\text{GP_ROTATION} = \begin{pmatrix} 0 & r_v & -q_v \\ -r_v & 0 & p_v \\ q_v & -p_v & 0 \end{pmatrix}$$

✓ **CALCULATE NEW VALUES OF ATTITUDE, VELOCITY, AND ALTITUDE**

The attitude, velocity, and altitude are each calculated by:

1. finding a rate of change from known values, and then
2. integrating this rate of change through one time step by some method of integration providing the accuracy specified. **That is:**

$$X_t = X_{t-1} + \int_{-1}^t \dot{X} dt$$

where \dot{X} represents the rate of change of velocity, altitude, or attitude. These are calculated according to the following formula:

$$\frac{d}{dt}(\text{variable}) = \alpha \times \text{variable} + \beta + \text{correction term}$$

Table 5.8 shows the values of the variables, α , β , and the correction terms **for each of the variables GP_ATTITUDE, GP_VELOCITY, and GP_ALTITUDE.**

- **Solve for the current values of GP_ATTITUDE, GP_VELOCITY, and GP_ALTITUDE using the equations given above, Table 5.8, and an appropriate integration method (see Appendix C Numerical Integration Instructions).³⁹**

Table 5.8: DIFFERENTIAL EQUATIONS

Variable	α	β	Correction Term
GP_ATTITUDE	GP_ROTATION	0	0
GP_VELOCITY	GP_ROTATION	GRAVITY * GP_ATTITUDE(i,3) + A_ACCELERATION i goes from 1 to 3	$K_MATRIX \times$ (TDLR_VELOCITY - GP_VELOCITY)

GP_ALTITUDE	0	$-GP_ATTITUDE \times$ $GP_VELOCITY$	$K_ALT \cdot (AR_ALTITUDE -$ $GP_ALTITUDE)$
-------------	---	--	---

In Table 5.8, note that:

1. Gravity is given as a scalar although it is actually a vector quantity. To obtain the correct quantity, the scalar given should be multiplied by the last column of the GP_ATTITUDE matrix to produce a column vector appropriate to the equation.
2. The equation for rate of change of altitude uses GP_ATTITUDE and GP_VELOCITY. The third column of GP_ATTITUDE should be treated as a row for this calculation. Thus element (1,3) of GP_ATTITUDE becomes the first element in a vector of one row and three columns. The element (2,3) becomes the second element, and (3,3) is the third element in this vector. This row-vector is then multiplied by the column-vector GP_VELOCITY to produce a scalar.

The correction terms represent a difference between the guidance processors value and the radar's value. The correction term is turned on or off by the "K" terms which are determined in the respective radar processors.

✓ DETERMINE IF ENGINES SHOULD BE ON OR OFF⁴⁰

Note that RE_SWITCH is initialized to on, while AE_SWITCH is initialized to off, and FRAME_ENGINES_IGNITED is initialized to zero by INIT_GCS. Use Table 5.9 to determine whether to turn axial engines on (set AE_SWITCH to on and set FRAME_ENGINES_IGNITED) or whether to turn axial and roll engines off (set AE_SWITCH and RE_SWITCH to off).

TABLE 5.9: DETERMINATION OF AXIAL AND ROLL ENGINE ON/OFF SWITCHES⁴¹

CURRENT STATE					ACTIONS		
AE_SWITCH	GP_ALTITUDE	$\sqrt{2 \cdot \text{GRAVITY} \cdot \text{GP_ALTITUDE}} +$ <i>x component of GP_VELOCITY</i> $\leq \text{MAX_NORMAL_VELOCIT}$?	Have engines been turned off in a prior frame?	TD_SENSED	FRAME_ENGINES_IGNITED	AE_SWITCH	RE_SWITCH
off	$\leq \text{ENGINES_ON_ALTITUDE}$	d	no	not sensed	current FRAME_COUNTER	on	
on	$\leq \text{DROP_HEIGHT}$	yes	d	not sensed		off	off
on	d	d	d	sensed		off	off

Note: A blank box under "ACTIONS" indicates no action is to be taken

"d" = don't care condition

✓ DETERMINE VELOCITY ERROR

The velocity error represents the difference between the x component of the velocity of the craft and the optimal velocity of the craft at the vehicle altitude (Shown in Figure 5.1). This distance is actually a difference between two velocities and is called VELOCITY_ERROR. The velocity-altitude contour is contained in two variables: CONTOUR_ALTITUDE and CONTOUR_VELOCITY. These are both arrays with 100 elements that contain known points along the contour. **CONTOUR_VELOCITY and CONTOUR_ALTITUDE are related such that element i of CONTOUR_VELOCITY is the optimum velocity at the altitude given by element i of CONTOUR_ALTITUDE.** It should be noted that the point in the first element is the lowest altitude given; and, as the index number increases, altitude increases. Since not all of these array elements may be needed, all unused elements beyond the highest given altitude will be filled with zeroes, and that the value of zero is never given for altitude except as this filler. The value of velocity at any other point may be found by linear interpolation (or extrapolation if the value is outside the range of the supplied contour) at the given vehicle altitude.

- **The optimal velocity** should be calculated by finding the present altitude in CONTOUR_ALTITUDE and then locating the corresponding velocity in CONTOUR_VELOCITY, using interpolation if necessary. **Let "optimal_velocity" represent the interpolated value calculated from the CONTOUR_VELOCITY table.**
- **Calculate VELOCITY_ERROR as follows:**⁴²

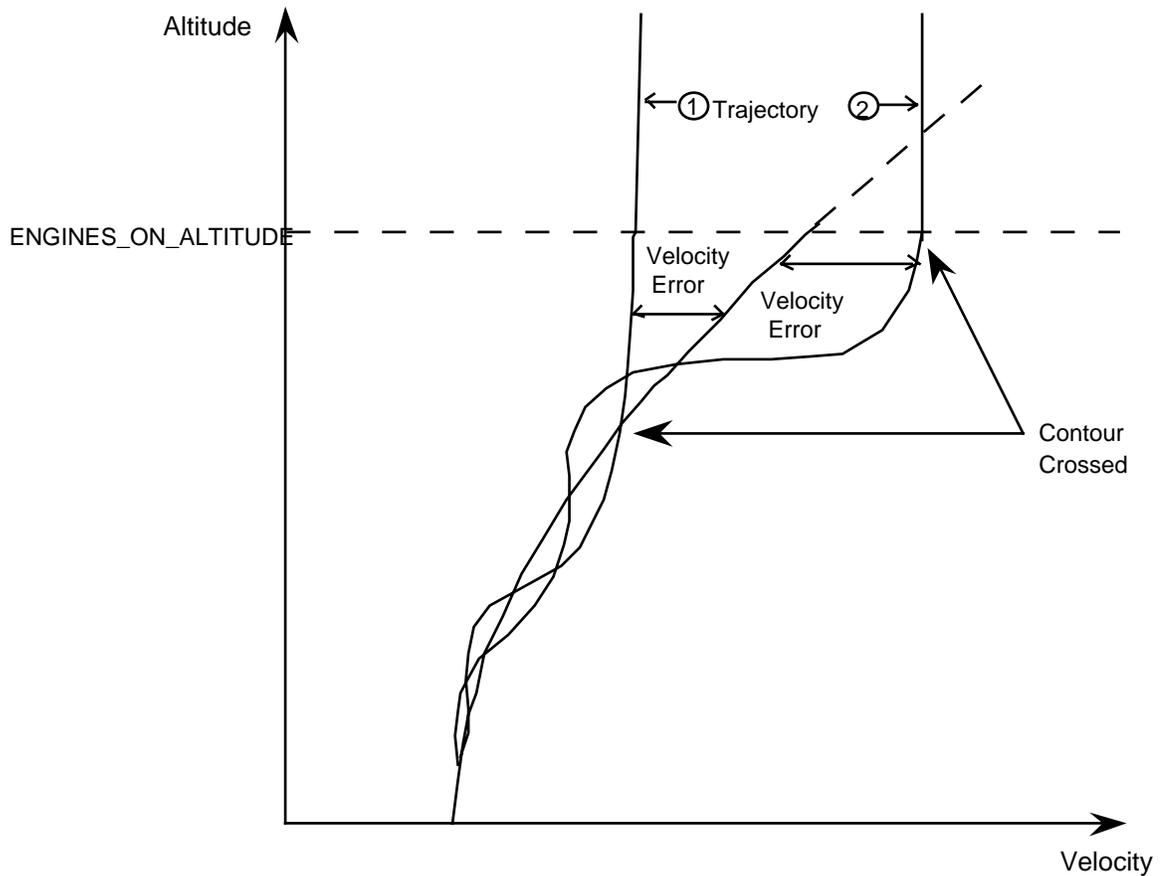
$$\text{VELOCITY_ERROR} = \text{x component of GP_VELOCITY} - \text{optimal_velocity}$$

✓ DETERMINE IF CONTOUR HAS BEEN CROSSED

- **If GP_ALTITUDE ≤ ENGINES_ON_ALTITUDE, then check whether the contour has been crossed as follows:**
- **If CONTOUR_CROSSED = "contour not crossed" and VELOCITY_ERROR ≥ 0, then set CONTOUR_CROSSED to "contour crossed". Otherwise CONTOUR_CROSSED should remain unchanged.**⁴³

Figure 5.1 shows two possible trajectories, with the point along each where the contour is first sensed and also an example of VELOCITY_ERROR. Note: the altitude where the engines are turned on should be the earliest point to check crossing the contour, even though the trajectory may have crossed the contour at some greater altitude.

Figure 5.1: VELOCITY-ALTITUDE CONTOUR⁴⁴



✓ DETERMINE GUIDANCE PHASE

- The guidance phase (GP_PHASE) is determined according to the events in Table 5.10. These phases are based upon information that may be provided by processes other than the guidance processor.

The current phase (GP_PHASE) and the event are to be used where appropriate to reset GP_PHASE to the next phase. If there is no combination of current phase and event from the table that is true, then GP_PHASE should not be changed. Note that the two columns labeled "PRESENT STATE" DESCRIPTION and "NEXT STATE DESCRIPTION" are for informational purposes only, and are not used in the setting of GP_PHASE.

Table 5.10: DETERMINATION OF GUIDANCE PHASE⁴⁶

CURRENT STATE			NEXT STATE	
GP_PHASE	CURRENT STATE DESCRIPTION	EVENT	ACTION GP_PHASE	NEXT STATE DESCRIPTION
1	Chute attached Engines off Touch Down not sensed	Altitude for turning engines on is sensed	2	Chute attached Engines on Touch down not sensed
2	Chute attached Engines on Touch down not sensed	Axial Engines become hot and the chute is released	3	Chute released Axial Engines Hot Touch down not sensed
2	Chute attached Engines on Touch down not sensed	Touched down is sensed	5	Chute attached Engines off Touch down sensed
3	Chute released Axial Engines Hot Touch down not sensed	Altitude \leq DROP_HEIGHT and TDS_STATUS = healthy and Touch down not sensed and <div style="border: 1px solid black; padding: 5px; width: fit-content; margin: 0 auto;"> $(\sqrt{2 \cdot \text{GRAVITY} \cdot \text{GP_ALTITUDE}} + x \text{ component of GP_VELOCITY}) \leq \text{MAX_NORMAL_VELOCITY}$ </div>	4	Chute released Engines off Touch down not sensed
3	Chute released Axial Engines Hot Touch down not sensed	Altitude \leq DROP_HEIGHT and TDS_STATUS = failed	5	Chute released Engines off Touch down not sensed
3	Chute released Axial Engines Hot Touch down not sensed	Touch down is sensed	5	Chute released Engines off Touch down sensed
4	Chute released Engines off Touch down not sensed	Touch down is sensed	5	Chute released Engines off Touch down sensed
4	Chute released Engines off Touch down not sensed	TDS_STATUS = failed	5	Chute released Engines off Touch down not sensed

- PHASE 1: If the altitude provided by the guidance processor is less than or equal to the ENGINES_ON_ALTITUDE, set GP_PHASE = 2.⁴⁷
- PHASE 2: If the axial engines have become hot and the parachute has been released, set GP_PHASE = 3. If touch down is sensed, set GP_PHASE = 5.⁴⁸
- PHASE 3: If touch down has not been sensed and DROP_HEIGHT has not been reached, then control the axial and roll engines to cause the lander to follow a gravity-turn steering descent. If

DROP_HEIGHT is reached **and touch down is not sensed and**

$$\sqrt{2 \cdot \text{GRAVITY} \cdot \text{GP_ALTITUDE}} + x \text{ component of GP_VELOCITY} \leq \text{MAX_NORMAL_VELOCITY}$$

and TDS_STATUS = healthy, then set GP_PHASE = 4. If DROP_HEIGHT is reached, and TDS_STATUS = failed, **then set GP_PHASE = 5**. If touch down is sensed, then **set GP_PHASE = 5**.⁴⁹

- PHASE 4 : If touch down has not been sensed and TDS_STATUS is healthy, **then take no action**. If TDS_STATUS is failed, **then set GP_PHASE to 5**. If touch down has been sensed, **set GP_PHASE to 5**.⁵⁰

✓ **DETERMINE WHICH SET OF CONTROL LAW PARAMETERS TO USE**⁵¹

The "Control Law Parameters" are a subset of the variables in the global data store named "RUN_PARAMETERS." This subset consists of the following variables: GVEI, GV, GVI, GR, GW, GWI, GQ, PE_MIN, PE_MAX, TE_MIN, TE_MAX, YE_MIN, and YE_MAX. Note that each one of these variables is an array of two elements. The elements with a subscript of one will be referred to as the "first" set of Control Law Parameters, while the elements with a subscript of two will be referred to as the "second" set of Control Law Parameters.

The variable CL is used to control which set of Control Law Parameters is used in the control laws at any given time by the functional unit AECLP. The functional unit GP must determine the value of CL for use by AECLP. The variable CL has two valid values, namely "first" which means that the first set of Control Law Parameters should be used by AECLP, and "second" which means that the second set of Control Law Parameters should be used by AECLP in the

equations for P_e , Y_e , P_e^L , Y_e^L , and TE_LIMIT. See the Data Requirements Dictionary for the actual numeric values for CL which correspond to "first" and "second." The variable CL is initialized to the value "first" by INIT_GCS, and thus the first set of parameters will be used by AECLP until CL is changed. The second set of Control Law Parameters should be used by AECLP at the first point where the lander crosses the constant-velocity part of the Velocity-Altitude contour. The constant-velocity part of the contour is the final part of the contour where the optimal velocity for the lander remains constant (and equal to DROP_SPEED). The GUIDANCE PROCESSOR (GP) must determine when to begin using the second set of Control Law Parameters, as follows:

- **If the following conditions are true:**
 - CL = first, and**
 - optimal_velocity = DROP_SPEED, and**
 - x component of GP_VELOCITY < DROP_SPEED**

Then

Set CL = second

Set TE_INTEGRAL = 0.0

GSP -- Gyroscope Sensor Processing (P-Spec 2.1.4)

PURPOSE Three fiber-optic ring gyroscopes are located on the lander, one for each of the x , y , and z axes as shown. The Gyroscope Sensor Processing (GSP) **functional unit** provides a measure of the vehicle's rotation rates through the conversion and filtering of the raw gyroscope data.

INPUT

ATMOSPHERIC_TEMP	G3
G4	G_COUNTER
G_GAIN_0	G_OFFSET
G_ROTATION	

OUTPUT

G_ROTATION	G_STATUS
------------	----------

PROCESS The output from each of the gyroscopes is a 16-bit quantity ($G_COUNTER$) divided into 2 parts: the lower 14 bits represent the vehicle's rate of rotation about that axis and the high-order bit represents the direction of this rotation. This is a sign-magnitude representation of the counter value that only uses the lower 14 bits of the magnitude portion of the number. Following is a map of $G_COUNTER$:

16	15	14	13	12	...	1
D	X	MAGNITUDE				

where D = direction, and X = unused. The high bit set to 1 indicates a negative rotation consistent with a right-handed coordinate system.

✓ ROTATE VARIABLES⁵²

- Rotate $G_ROTATION$.

✓ ADJUST GAIN

The standard gain (G_GAIN_0) must be adjusted for the effects of temperature prior to the conversion of the raw gyroscope values. The adjusted gain is a quadratic function of the ambient temperature ($ATMOSPHERIC_TEMP$) and the standard gain.

That is,

$$G_GAIN(i) = G_GAIN_0(i) + (G3 \cdot ATMOSPHERIC_TEMP) + (G4 \cdot ATMOSPHERIC_TEMP^2)$$

where i ranges from 1 to 3 and represents the three directions x , y , and z .

✓ CONVERT G_COUNTER

The rotation rate is linear with respect to the unprocessed gyroscope values, i.e. the lower 14 bits must be converted. G_GAIN is the multiplier for this conversion and G_OFFSET is the constant offset. The equation for converting counter to rotation then becomes:

$$G_ROTATION(i) = G_OFFSET(i) + G_GAIN(i) * (G_COUNTER(i))$$

where i ranges from 1 to 3 and represents the three directions x, y, and z.

✓ SET GYROSCOPE STATUS TO HEALTHY.

- **Set G_STATUS to healthy.**

RECLP -- Roll Engine Control Law Processing (P-Spec 2.3.2)

PURPOSE RECLP generates the roll engine command which controls the firing pulse and direction of the roll engines.

INPUT

DELTA_T	G_ROTATION
P1	P2
P3	P4
RE_SWITCH	THETA
THETA1	THETA2

OUTPUT

RE_CMD	RE_STATUS
THETA	

PROCESS Roll control of the lander is achieved by generating the **roll** commands as functions of the **differences between the actual and desirable values for the roll angle and rate. These differences are limited, and the control commands are proportional to them.** Note that once the roll command (RE_CMD) has been set with the correct value, it will automatically be sent to the engines during the next call to GCS_SIM_RENDEZVOUS. **The steps to be performed are as follows:**⁵³

- ✓ DETERMINE IF ENGINES ARE ON⁵⁴
 - If RE_SWITCH is off, then **set RE_CMD to 1, and proceed directly to the step "SET ROLL ENGINE STATUS TO HEALTHY."**

- ✓ DETERMINE PULSE INTENSITY AND DIRECTION
 - The pulse intensity and direction **are** derived from the graph shown in Figure 5.2 using $(p_v)_t$. For each region of the graph, the intensity is given, followed by the direction inside parentheses. Note that the x axis represents the integral of the roll rate. This is really the present angle of roll. This integral should be calculated by Euler's method (see **Appendix C**). As an example, THETA = THETA + (integral of roll **rate** for this step). **The variable THETA will be initialized by INIT_GCS.** Note that when the vehicle status is located on a boundary between two or more roll command regions, the lowest intensity signal should be used to avoid over-commanding the engines. **One should refer to the Data Requirements Dictionary under RE_CMD for the actual values for intensity and direction.**

✓ DETERMINE ROLL ENGINE COMMAND

- The pulse intensity and direction **are** packed into the lowest three lower-order bits of the actual roll engine command (RE_CMD) as shown:

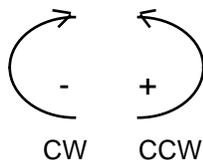
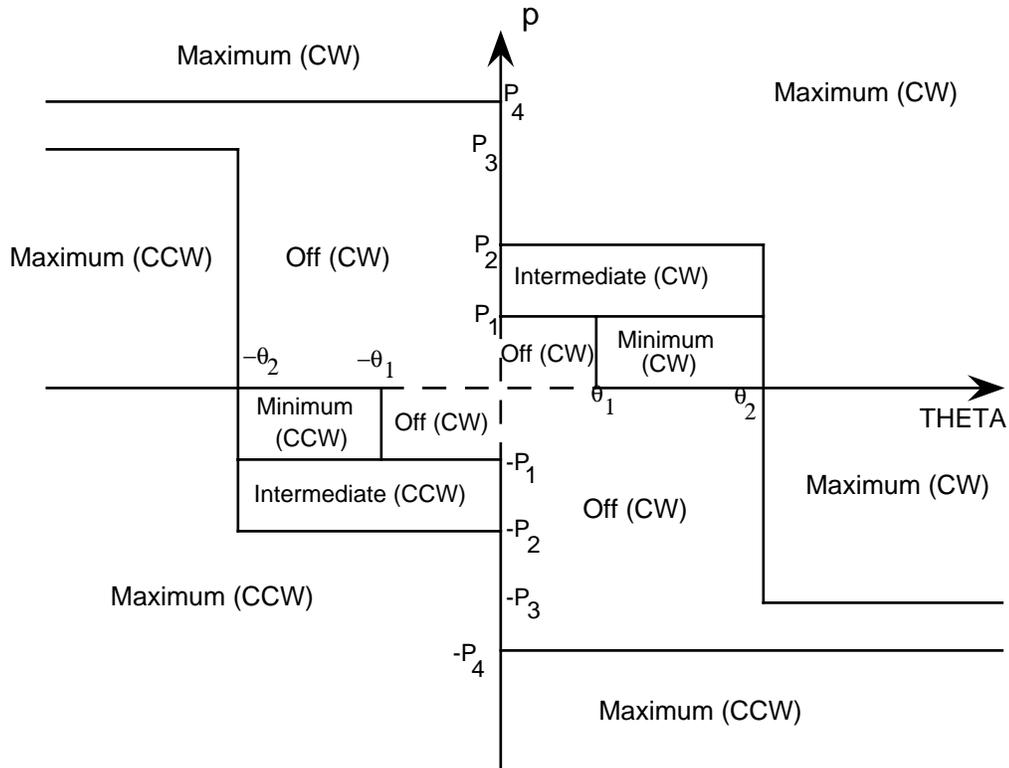
X	X	X	☺	X	I	I	D
16	15	14	☹	4	3	2	1

where X = unused, I = intensity, and D = direction. The bits marked "X = unused" in RE_CMD must be left at 0.

✓ SET ROLL ENGINE STATUS TO HEALTHY

- Set RE_STATUS to healthy.

Figure 5.2: GRAPH FOR DERIVING ROLL ENGINE COMMANDS⁵⁵



CW = Clockwise
CCW = Counterclockwise

Note: Off, Minimum, Intermediate, and Maximum are Intensities
CW, CCW are Directions

TDLRSP -- Touch Down Landing Radar Sensor Processing (P-Spec 2.1.3)

PURPOSE A single touch down landing radar (TDLR) gauges the velocity of the vehicle during terminal descent. This radar is a doppler radar with four radar beams, each of which emanates from the vehicle's center of gravity with a slight offset from the vehicle's \vec{x}_v axis. The radar beams form the edges of the pyramid as shown in Figure 5.3

The Touch Down Landing Radar Sensor Processing (TDLRSP) **functional unit** converts measurements of the frequency shift of each beams reflection into vehicle velocities; **however**, the receivers associated with each beam may not find a usable reflection. If no usable reflection is found, the receiver returns a status of beam in search mode (**unlocked**).

INPUT

DELTA_T	FRAME_BEAM_UNLOCKED
FRAME_COUNTER	K_MATRIX
TDLR_ANGLES	TDLR_COUNTER
TDLR_GAIN	TDLR_LOCK_TIME
TDLR_OFFSET	TDLR_STATE
TDLR_VELOCITY	

OUTPUT

FRAME_BEAM_UNLOCKED	K_MATRIX
TDLR_STATE	TDLR_STATUS
TDLR_VELOCITY	

PROCESS It is only necessary that this functional unit perform its normal calculations every other frame, namely on the odd-numbered frames; however, one will notice that in the scheduling Table 4.1, it is required that this functional unit execute every frame. The reason for this is that during its normal processing it must rotate history variables. This means that during the frames when it does not need to calculate new outputs, namely the even-numbered frames, it must still rotate its history variables and set its new or current values equal to the previous values, thus creating double entries for each rotated variable. By doubling the entries, consistency of time histories will be maintained at the expense of keeping two copies of each value in these variables, and forcing the functional unit to execute every frame.⁵⁶

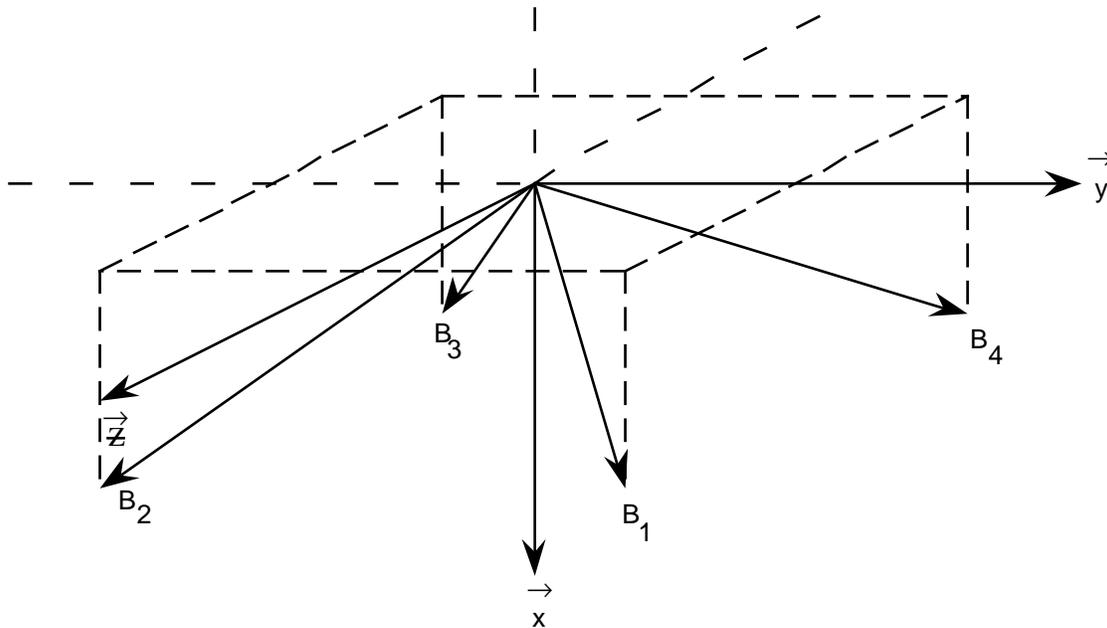
The value returned by each beam (TDLR_COUNTER) is proportional to the beam frequency shift down that beam, which is, in turn, proportional to the velocity down that beam. The processing of the TDLR_COUNTER data into the component velocities along the vehicle's \vec{x} , \vec{y} , and \vec{z} axes requires the following steps:

- ✓ ROTATE VARIABLES⁵⁷
 - Rotate TDLR_VELOCITY and K-MATRIX.

✓ **PERFORM ALTERNATE PROCESSING IF THIS IS AN EVEN-NUMBERED FRAME**⁵⁸

- **If FRAME_COUNTER is an even number, then do the following::**
 - **Insure that the values for the current TDLR_VELOCITY array are equal to the values for the previous TDLR_VELOCITY array and also that the values for the current K_MATRIX array are equal to the values for the previous K_MATRIX array.**
 - **Exit from this functional unit.**

Figure 5.3: DOPPLER RADAR BEAM LOCATIONS



✓ **DETERMINE RADAR BEAM STATES**⁵⁹

The processing of the four radar beams depends on the current state of the radar, i.e. whether or not each of the four beams is searching or in lock, **and also upon the previous states of the beams**. Note that at the beginning of each trajectory, FRAME_BEAM_UNLOCKED will be set to zero, thus meaning that the beam has never been unlocked. If the receiver for a beam does not sense an echo (i.e. the beam is in search mode), the corresponding TDLR_COUNTER value will be zero. Note that a beam which becomes unlocked will be ignored for TDLR_LOCK_TIME seconds.

- **Use Table 5.11 to determine the state (TDLR_STATE and FRAME_BEAM_UNLOCKED) for each of the four beams.**

Table 5.11: DETERMINATION OF RADAR BEAM STATES⁶⁰

CURRENT STATE			ACTIONS	
TDLR_STATE	TDLR_COUNTER	$DELTA_T$ $\cdot (FRAME_COUNTER - FRAME_BEAM_UNLOCKE$ $\geq TDLR_LOCK_TIME?$	TDLR_STATE	FRAME_BEAM_UNLOCKED
locked	0	d	unlocked	current FRAME_COUNTER
unlocked	$\neq 0$	yes	locked	
unlocked	0	yes		current FRAME_COUNTER

Note: A blank box under "ACTIONS" indicates no action is to be taken
"d" = don't care condition

✓ DETERMINE BEAM VELOCITIES

A beam velocity is a linear function of its TDLR_COUNTER value where the gain (TDLR_GAIN) specifies the slope and the offset (TDLR_OFFSET) specifies the intercept.

- Calculate the beam velocities as follows:

$$\mathbf{B}(i) = TDLR_OFFSET + TDLR_GAIN * (TDLR_COUNTER(i))$$

where i ranges from 1 to 4 and represents the four radar beams.

✓ PROCESS THE BEAM VELOCITIES⁶¹

- Use Table 5.12 to calculate values for \hat{B}_x , \hat{B}_y , and \hat{B}_z , which are the processed beam velocities. Note that in Table 5.12, B_i is shorthand for $B(i)$, where i ranges from 1 to 4. Note also that the knowledge of which beams are in lock is used to determine

which line of the table to use in order to calculate \hat{B}_x , \hat{B}_y , and \hat{B}_z .

✓ CONVERT TO BODY VELOCITIES⁶²

- In order to convert the processed beam velocities to body velocities (TDLR_VELOCITY), use the following equations, which make use of the angles α , β and γ (TDLR_ANGLES) which are the offsets of the beams from the body axes:

$$TDLR_VELOCITY(1) = \frac{\hat{B}_y}{\cos \alpha}$$

$$TDLR_VELOCITY(2) = \frac{\hat{B}_v}{\cos \beta}$$

$$TDLR_VELOCITY(3) = \frac{\hat{B}_v}{\cos \gamma}$$

✓ SET VALUES IN K_MATRIX⁶³

When calculating the vehicle velocity, the Guidance Processor must know which components of the body velocities are usable. A value of one in the diagonal element of the K_MATRIX indicates that the corresponding velocity should be used, while a value of zero indicates that it should not.

- Use Table 5.12 to set the values for K_x , K_y , and K_z in K_MATRIX, (again on the basis of which beams are in lock), as follows:

$$K_MATRIX = \begin{pmatrix} K_x & 0 & 0 \\ 0 & K_y & 0 \\ 0 & 0 & K_z \end{pmatrix}$$

✓ SET TDLR_STATUS

- Set **all elements** of TDLR_STATUS to healthy.

Table 5.12: PROCESSING OF DOPPLER RADAR BEAMS IN LOCK⁶⁴

BEAMS IN LOCK	\hat{B}_x	K_x	\hat{B}_y	K_y	\hat{B}_z	K_z
none	0	0	0	0	0	0
B_1	0	0	0	0	0	0
B_2	0	0	0	0	0	0
B_3	0	0	0	0	0	0
B_4	0	0	0	0	0	0
B_1, B_2	0	0	$(B_1 - B_2)/2$	1	0	0
B_1, B_3	$(B_1 + B_3)/2$	1	0	0	0	0
B_1, B_4	0	0	0	0	$(B_1 - B_4)/2$	1
B_2, B_3	0	0	0	0	$(B_2 - B_3)/2$	1
B_2, B_4	$(B_2 + B_4)/2$	1	0	0	0	0
B_3, B_4	0	0	$(B_4 - B_3)/2$	1	0	0
B_1, B_2, B_3	$(B_1 + B_3)/2$	1	$(B_1 - B_2)/2$	1	$(B_2 - B_3)/2$	1
B_1, B_2, B_4	$(B_2 + B_4)/2$	1	$(B_1 - B_2)/2$	1	$(B_1 - B_4)/2$	1
B_1, B_3, B_4	$(B_1 + B_3)/2$	1	$(B_4 - B_3)/2$	1	$(B_1 - B_4)/2$	1
B_2, B_3, B_4	$(B_2 + B_4)/2$	1	$(B_4 - B_3)/2$	1	$(B_2 - B_3)/2$	1
B_1, B_2, B_3, B_4	$(B_1 + B_2 + B_3 + B_4)/4$	1	$(B_1 - B_2 - B_3 + B_4)/4$	1	$(B_1 + B_2 - B_3 - B_4)/4$	1

TDSP -- Touch Down Sensor Processing (P-Spec 2.1.6)

PURPOSE The touch down sensor is attached to the end of a rod which is attached to the bottom of the vehicle. Its purpose is to trigger engine shutdown when the vehicle is at the correct distance from the surface. This shutdown is necessary to:

- avoid the stirring up of dust and debris and
- avoid scorching immediate area of the experiment site.

INPUT

TD_COUNTER	TDS_STATUS
------------	------------

OUTPUT

TD_SENSED	TDS_STATUS
-----------	------------

PROCESS The touch down sensor is a simple switch at the end of a pole on the underside of the lander. **If the sensor is functioning properly, then TD_COUNTER will contain one of only two 16-bit values, namely sixteen "ones", which means that touch down has been sensed, or sixteen "zeroes", which means that touch down has not been sensed. If the sensor has failed due to electrical noise, TD_COUNTER will contain some combination of "ones" and "zeroes" other than all "ones" or all "zeroes".**⁶⁵

- ✓ DETERMINE STATUS OF TOUCH DOWN SENSOR AND WHETHER TOUCH DOWN HAS BEEN SENSED:
 - Use Table 5.13 to determine whether the touch down sensor is functioning properly (set TDS_STATUS), and whether touch down has been sensed (set TD_SENSED). Note that if the sensor fails, the guidance processor will decide when the vehicle has touched down.⁶⁶

Table 5.13: DETERMINATION OF TOUCH DOWN SENSOR AND STATUS⁶⁷

CURRENT STATE		ACTIONS	
TDS_STATUS	TD_COUNTER	TD_SENSED	TDS_STATUS
healthy	all zeroes	not sensed	
healthy	all ones	sensed	
healthy	mixture of ones & zeroes	not sensed	failed

Note: A blank block under "ACTIONS" indicates no action is to be taken

TSP -- Temperature Sensor Processing (P-Spec 2.1.5)

PURPOSE A temperature gauge on the vehicle is used to adjust the response of the accelerometers and gyroscopes. The gauge contains two temperature sensing devices, namely a solid-state sensor and a matched pair of thermocouples. The Temperature Sensor Processing (TSP) **functional unit** determines the ambient temperature, using either the solid-state sensor or the thermocouple pair in a manner maximizing the accuracy of the measurement.

INPUT

M1	M2
M3	M4
SS_TEMP	T1
T2	T3
T4	THERMO_TEMP

OUTPUT

ATMOSPHERIC_TEMP	TS_STATUS
------------------	-----------

PROCESS The temperature values from the solid-state sensor are highly quantized. The processing of raw temperature data from the solid-state sensor and thermocouple pair, SS_TEMP and THERMO_TEMP, is based on the solid-state sensor being less accurate than the thermocouple pair, but having a greater usable operating range.

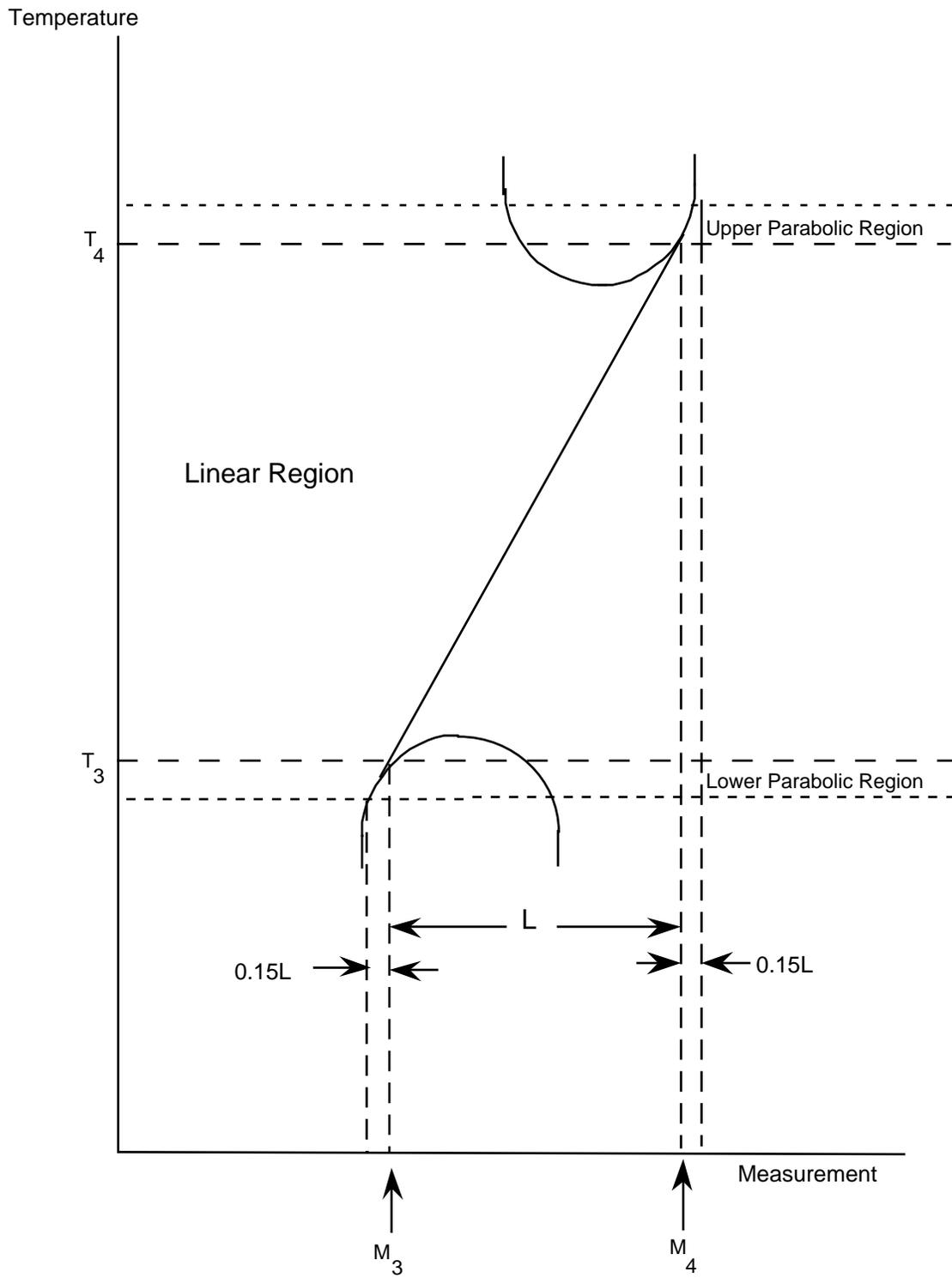
The ambient temperature (ATMOSPHERIC_TEMP) is to be calculated using either the solid state sensor value (SS_TEMP) or the thermocouple sensor value (THERMO_TEMP). Since the thermocouple sensor is more accurate, it should be used whenever possible; the solid state sensor should be used only if the temperature does not lie within the usable range of the thermocouple pair. ⁶⁹

The response of the solid-state temperature sensor is linear with respect to the ambient temperature and is computed using the two calibration points (M1, T1) and (M2, T2) which characterize the line.

The response of the thermocouple pair is calibrated differently depending on the region (linear or parabolic) where the measurement lies (see Figure 5.4):

Thermocouple linear region - The linear region is bounded by the calibration points used by the thermocouple sensor (i.e., [M3, T3] and [M4, T4] inclusive). Temperatures measured within this region are calibrated accordingly.

Figure 5.4: CALIBRATION OF THERMOCOUPLE PAIR⁷⁰



Thermocouple parabolic regions - The upper and lower parabolic regions extend plus or minus 15 percent of the difference between the measured calibration points, M4 and M3, respectively. These parabolic regions each intersect the line at the calibration points. The rate of change in temperature, with respect to the thermocouple measurements, is continuous at these intersections. The upper (and lower) parabolas are defined so that the temperature goes up (or down) as the square of the measurement value (THERMO_TEMP). The parabolas are offset along both the temperature and measurement axes. By using the values of T3, T4, M3, and M4, and the fact that the function is continuous at the endpoints, the offsets for the parabolas may be determined, and the equations for the parabolas may be generated. **Note that the line in the linear region in Figure 5.4 is tangent to both parabolas.**

The processing of the values SS_TEMP and THERMO_TEMP into an accurate measure of ambient temperature (ATMOSPHERIC_TEMP) requires several steps, as follows:

- ✓ **CALCULATE THE SOLID STATE TEMPERATURE⁷¹**
 - Use the value of SS_TEMP and the equation appropriate to the solid-state linear region to compute the temperature.

- ✓ **DETERMINE WHETHER TO USE SOLID STATE OR THERMOCOUPLE TEMPERATURE⁷²**
 - If the temperature derived from SS_TEMP in the previous step does not fall within the accurate temperature response zone of the thermocouple pair (the linear as well as parabolic regions), then set ATMOSPHERIC_TEMP to the temperature derived from SS_TEMP and proceed directly to the step labeled "SET STATUS TO HEALTHY"; otherwise, proceed to the step "CALCULATE THE THERMOCOUPLE TEMPERATURE".

- ✓ **CALCULATE THE THERMOCOUPLE TEMPERATURE⁷³**
 - Use the value of THERMO_TEMP to determine whether the temperature lies in the thermocouple linear or the upper parabolic or the lower parabolic region.
 - Use the value of THERMO_TEMP and the equation appropriate to the particular thermocouple region (as determined above) to calculate ATMOSPHERIC_TEMP.

- ✓ **SET STATUS TO HEALTHY**
 - Set the values of both elements of TS_STATUS to healthy.

6. DATA REQUIREMENTS DICTIONARY

PART I. DATA ELEMENT DESCRIPTIONS

The following template has been constructed for defining the data elements **in the four required global data stores and the optional variables shown in Table 6.5:**

NAME: DESCRIPTION: USED IN: UNITS: RANGE: DATA TYPE: ATTRIBUTE: DATA STORE LOCATION: ACCURACY:
--

NAME This field gives the name of the variable used in the specification. The variable name used during coding must be the same as specified.

DESCRIPTION This field gives a brief description of the variable.

USED IN This field provides a reference to the **functional units** using this variable.

UNITS This field indicates the unit of measure for the data contained in the variable being defined.

RANGE This field specifies the **acceptable** range of data values for the variable.

DATA TYPE The data type field specifies the data type to be used when declaring the variable during coding.

ATTRIBUTE This field indicates whether or not the variable contains data, control information, or a data condition.

DATA STORE LOCATION This field references the common region where the variable must be stored.

ACCURACY This field dictates the degree of accuracy required for output comparisons to be made **between implementations**. In the data dictionary, accuracy is listed as N/A where accuracy is not applicable, or TBD where accuracy is (T)o (B)e (D)etermined later. A formal modification will be released when the values of the accuracy requirements have been approved.

NAME: A_ACCELERATION⁷⁴
DESCRIPTION: vehicle accelerations
USED IN: AECLP, ASP, CP, GP

ACCURACY: N/A

UNITS: $\frac{meters}{sec^2}$

RANGE: [-20, 5]

DATA TYPE: array (1..3, 0..4) of real*8

ATTRIBUTE: data

DATA STORE LOCATION: SENSOR_OUTPUT

ACCURACY: TBD

NAME: A_BIAS

DESCRIPTION: characteristic bias in the
accelerometer measurements

USED IN: ASP

UNITS: $\frac{meters}{sec^2}$

RANGE: [-30, 0]

DATA TYPE: array (1..3) of real*8

ATTRIBUTE: data

DATA STORE LOCATION: RUN_PARAMETERS

ACCURACY: N/A

NAME: A_COUNTER

DESCRIPTION: accelerations along the \bar{x} , \bar{y} , and
 \bar{z} axes

USED IN: ASP

UNITS: none

RANGE: [0, 2¹⁵ -1]

DATA TYPE: array (1..3) of Integer*2

ATTRIBUTE: data

DATA STORE LOCATION: EXTERNAL

ACCURACY: N/A

NAME: A_GAIN_0⁷⁵

DESCRIPTION: standard gain in the accelerations

USED IN: ASP

UNITS: $\frac{meters}{sec^2}$

RANGE: [0, 1]

DATA TYPE: array (1..3) of real*8

ATTRIBUTE: data

DATA STORE LOCATION: RUN_PARAMETERS

ACCURACY: N/A

NAME: A_SCALE⁷⁶

DESCRIPTION: multiplicative constant used to
determine limit on deviation accelerometer values.

USED IN: ASP

UNITS: none

RANGE: [0, 3]

DATA TYPE: Integer*4

ATTRIBUTE: data

DATA STORE LOCATION: RUN_PARAMETERS

NAME: A_STATUS
DESCRIPTION: Flag indicating whether or not the accelerometers are working properly.
USED IN: ASP, CP
UNITS: none
RANGE: [0 : healthy, 1: unhealthy]
DATA TYPE: array (1..3, 0..3) of logical*1
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: N/A

NAME: AECLP_DONE
DESCRIPTION: Flag indicating completion of AECLP task.
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [FALSE: running of task AECLP incomplete, TRUE: running of task AECLP complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: AE_CMD
DESCRIPTION: Valve settings for the axial engines.
USED IN: AECLP, CP
UNITS: none
RANGE: [0, 127]
DATA TYPE: array (1..3) of Integer*2
ATTRIBUTE: data
DATA STORE LOCATION: EXTERNAL
ACCURACY: TBD

NAME: AE_STATUS
DESCRIPTION: **Status of axial engines.**
USED IN: AECLP, CP
UNITS: none
RANGE: [0: Healthy, 1: Failed.]
DATA TYPE: logical*1
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: N/A

NAME: AE_SWITCH
DESCRIPTION: Flag indicating whether or not axial engines are turned on.
USED IN: AECLP, GP
UNITS: none
RANGE: [0: axial engines are off, 1: axial engines are on.]
DATA TYPE: logical*1
ATTRIBUTE: data condition
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: N/A

NAME: AE_TEMP
DESCRIPTION: Temperature of axial engines when they are turned on.
USED IN: AECLP, CP, CRCP, GP
UNITS: none
RANGE: [0: Cold, 1: Warming-Up, 2: Hot]
DATA TYPE: logical*1
ATTRIBUTE: data condition
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: N/A

NAME: ALPHA_MATRIX
DESCRIPTION: Matrix of misalignment angles
USED IN: ASP
UNITS: none
RANGE: $[-\pi, \pi]$
DATA TYPE: array (1..3, 1..3) of real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: AR_ALTITUDE
DESCRIPTION: altimeter radar height above terrain
USED IN: ARSP, CP, GP
UNITS: meters
RANGE: [0, 2000]
DATA TYPE: array (0..4) of real*8
ATTRIBUTE: data
DATA STORE LOCATION: SENSOR_OUTPUT
ACCURACY: TBD

NAME: AR_COUNTER⁷⁷
DESCRIPTION: counter containing elapsed time since transmission of radar pulse
USED IN: ARSP
UNITS: Cycles
RANGE: $[-1, 2^{15}-1]$
DATA TYPE: Integer*2
ATTRIBUTE: data
DATA STORE LOCATION: EXTERNAL
ACCURACY: N/A

NAME: AR_FREQUENCY⁷⁸
DESCRIPTION: increment frequency of AR_COUNTER
USED IN: ARSP
UNITS: $\frac{\text{cycles}}{\text{sec}}$
RANGE: $[1, 2.45 \times 10^9]$
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: AR_STATUS
DESCRIPTION: status of the altimeter radars
USED IN: ARSP, CP
UNITS: none
RANGE: [0 : healthy, 1: failed]
DATA TYPE: array (0..4) of logical*1
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: N/A

NAME: ARSP_DONE
DESCRIPTION: Flag indicating completion of ARSP task.
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [FALSE: running of task ARSP incomplete, TRUE: running of task ARSP complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: ASP_DONE
DESCRIPTION: Flag indicating completion of ASP task.
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [FALSE: running of task ASP incomplete, TRUE: running of task ASP complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: ATMOSPHERIC_TEMP⁷⁹
DESCRIPTION: atmospheric temperature
USED IN: ASP, CP, GSP, TSP
UNITS: degrees C
RANGE: [-200, 25]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: SENSOR_OUTPUT,
ACCURACY: TBD

NAME: C_STATUS
DESCRIPTION: Flag indicating whether or not the communications processor is working properly.
USED IN: CP
UNITS: none
RANGE: [0 : healthy, 1: failed]
DATA TYPE: logical*1
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: N/A

NAME: CHUTE_RELEASED
DESCRIPTION: signal indicating parachute has been released
USED IN: AECLP, CP, CRCP, GP
UNITS: none
RANGE: [0: Chute Attached, 1: Chute Released]
DATA TYPE: logical*1
ATTRIBUTE: data condition
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: N/A

NAME: CL⁸⁰
DESCRIPTION: Index which specifies which set of Control Law Parameters to use
USED IN: AECLP, GP
UNITS: none
RANGE: [1: first, 2: second]
DATA TYPE: Integer*2
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: N/A

NAME: CLP_DONE
DESCRIPTION: Control signal which indicates whether or not Control Law Processing function has completed.
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [FALSE: running of Control Law Processing function incomplete, TRUE: running of Control Law Processing function complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: COMM_SYNC_PATTERN
DESCRIPTION: sixteen bit synchronization pattern
USED IN: CP
UNITS: none
RANGE: [1101100110110010]
DATA TYPE: Integer*2
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: CONTOUR_ALTITUDE⁸¹
DESCRIPTION: Altitude in velocity-altitude contour.
USED IN: GP
UNITS: kilometers
RANGE: [-.01, 2]
DATA TYPE: array (1..100) of real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: CONTOUR_CROSSED⁸²
DESCRIPTION: Indicates if the velocity-altitude contour has been sensed.
USED IN: AECLP, CP, GP
UNITS: none
RANGE: [0: contour not **crossed**, 1: contour **crossed**]
DATA TYPE: logical*1
ATTRIBUTE: data condition
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: N/A

NAME: CONTOUR_VELOCITY
DESCRIPTION: Velocity in velocity-altitude contour.
USED IN: GP
UNITS: $\frac{\text{kilometers}}{\text{sec}}$
RANGE: [0, 0.5]
DATA TYPE: array (1..100) of real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: CP_DONE
DESCRIPTION: Flag indicating completion of CP task.
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [FALSE: running of task CP incomplete, TRUE: running of task CP complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: CRCP_DONE
DESCRIPTION: Flag indicating completion of CRCP task.
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [FALSE: running of task CRCP incomplete, TRUE: running of task CRCP complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: DELTA_T⁸³
DESCRIPTION: Time step duration.
USED IN: AECLP, GP, RECLP, TDLRSP
UNITS: seconds
RANGE: [0.005, 0.20]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS

ACCURACY: N/A

NAME: DROP_HEIGHT
DESCRIPTION: Height from which vehicle should free-fall to surface
USED IN: GP
UNITS: meters
RANGE: [0, 100]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: DROP_SPEED⁸⁴
DESCRIPTION: **Optimal speed during constant velocity descent.**
USED IN: GP
UNITS: $\frac{\text{meters}}{\text{sec}}$
RANGE: [0, 4.0]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: ENGINES_ON_ALTITUDE
DESCRIPTION: Altitude at which the axial engines are turned on.
USED IN: AECLP, GP
UNITS: meters
RANGE: [0, 2000]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: FRAME_BEAM_UNLOCKED
DESCRIPTION: Variable containing the number of the frame during which the radar beam unlocked
USED IN: TDLRSP
UNITS: none
RANGE: [0, 2³¹-1]
DATA TYPE: array (1..4) of Integer*4
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: TBD

NAME: FRAME_COUNTER
DESCRIPTION: Counter containing the number of the present frame
USED IN: AECLP, ARSP, CP, GP, TDLRSP
UNITS: none
RANGE: [1, 2³¹-1]
DATA TYPE: Integer*4
ATTRIBUTE: data

DATA STORE LOCATION: EXTERNAL
ACCURACY: N/A

NAME: FRAME_ENGINES_IGNITED
DESCRIPTION: Variable containing the number of the frame during which the engines were ignited
USED IN: AECLP, GP
UNITS: none
RANGE: [0, 2³¹-1]
DATA TYPE: Integer*4
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE
ACCURACY: TBD

NAME: FULL_UP_TIME
DESCRIPTION: Time for axial engines to reach optimum operational condition
USED IN: AECLP
UNITS: seconds
RANGE: [0, 60]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: G1
DESCRIPTION: coefficient used to adjust A_GAIN
USED IN: ASP
UNITS: $\frac{\text{meters}}{\text{deg } \text{ree } C^2}$
RANGE: [-5, 5]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: G2
DESCRIPTION: coefficient used to adjust A_GAIN
USED IN: ASP
UNITS: $\frac{\text{meters}}{\text{deg } \text{ree } C^2}$
RANGE: [-5, 5]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: G3
DESCRIPTION: coefficient used to adjust G_GAIN
USED IN: GSP
UNITS: $\frac{\text{radians}}{\text{deg } \text{ree } C}$

RANGE: [-5, 5]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A
NAME: G4
DESCRIPTION: coefficient used to adjust G_GAIN
USED IN: GSP
UNITS: $\frac{\text{radians}}{\text{deg } \text{ree } C^2}$

RANGE: [-5, 5]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: G_COUNTER
DESCRIPTION: gyroscope measurement of vehicle rotation rates
USED IN: GSP
UNITS: none
RANGE: [-(2¹⁴-1), 2¹⁴-1]
DATA TYPE: array (1..3) of Integer*2
ATTRIBUTE: data
DATA STORE LOCATION: EXTERNAL
ACCURACY: N/A

NAME: G_GAIN_0⁸⁵
DESCRIPTION: standard gain in vehicle rotation rates as measured by the gyroscopes
USED IN: GSP
UNITS: $\frac{\text{radians}}{\text{sec}}$
RANGE: [-1, 1]
DATA TYPE: array (1..3) of real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: G_OFFSET
DESCRIPTION: standard offset of the **rotation raw** values
USED IN: GSP
UNITS: $\frac{\text{radians}}{\text{sec}}$
RANGE: [-0.5, 0.5]
DATA TYPE: array (1..3) of real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: G_ROTATION⁸⁶
DESCRIPTION: vehicle rotation rates
USED IN: CP, GSP, GP, RECLP

UNITS: $\frac{\text{radians}}{\text{sec}}$
RANGE: [-1.0, 1.0]
DATA TYPE: array (1..3, 0..4)of real*8
ATTRIBUTE: data
DATA STORE LOCATION: SENSOR_OUTPUT
ACCURACY: TBD
NAME: G_STATUS
DESCRIPTION: status of the gyroscopes
USED IN: CP, GSP
UNITS: none
RANGE: [0 : healthy, 1: failed]
DATA TYPE: logical*1
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: N/A

NAME: GA
DESCRIPTION: gain
USED IN: AECLP
UNITS: $\frac{\text{sec}}{\text{meter}}$
RANGE: [0, 50]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GAX⁸⁷
DESCRIPTION: gain
USED IN: AECLP
UNITS: none
RANGE: [0, 5]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GP1
DESCRIPTION: gain
USED IN: AECLP
UNITS: none
RANGE: [-5, 5]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GP2
DESCRIPTION: gain
USED IN: AECLP
UNITS: none
RANGE: [-5, 5]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GP_ALTITUDE⁸⁸
DESCRIPTION: altitude as seen by guidance processor
USED IN: AECLP, CP, GP
UNITS: meters
RANGE: [0, 2000]
DATA TYPE: array (0..4) of real*8
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: TBD

NAME: GP_ATTITUDE
DESCRIPTION: direction cosine matrix
USED IN: AECLP, CP, GP
UNITS: none
RANGE: [-1, 1]
DATA TYPE: array (1..3, 1..3, 0..4) real*8
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: TBD

NAME: GP_DONE
DESCRIPTION: Flag indicating completion of GP task.
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [FALSE: running of task GP incomplete, TRUE: running of task GP complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: GP_PHASE⁸⁹
DESCRIPTION: phase of operation as seen by guidance processor
USED IN: CP, GP
UNITS: none
RANGE: [1, 5]
DATA TYPE: integer*4
ATTRIBUTE: data condition
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: TBD

NAME: GP_ROTATION⁹⁰
DESCRIPTION: rotation rates as determined by the guidance processing **functional unit**
USED IN: AECLP, CP, GP
UNITS: $\frac{\text{radians}}{\text{sec}}$
RANGE: [-1.0, 1.0]
DATA TYPE: array (1..3, 1..3) real*8
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE,
ACCURACY: TBD

NAME: GP_VELOCITY
DESCRIPTION: Velocity as corrected by the guidance algorithm.
USED IN: AECLP, CP, GP
UNITS: $\frac{\text{meters}}{\text{sec}}$
RANGE: [-100, 100]
DATA TYPE: array (1..3, 0..4) of real*8
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: TBD

NAME: GPY
DESCRIPTION: gain
USED IN: AECLP
UNITS: none
RANGE: [-5, 5]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GQ⁹¹
DESCRIPTION: gain
USED IN: AECLP
UNITS: seconds
RANGE: [-5, 8]
DATA TYPE: **array (1..2) of real*8**
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GR⁹²
DESCRIPTION: gain
USED IN: AECLP
UNITS: seconds
RANGE: [-5, 8]
DATA TYPE: **array (1..2) of real*8**
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GRAVITY
DESCRIPTION: gravity of planet
USED IN: AECLP, GP
UNITS: $\frac{\text{meters}}{\text{sec}^2}$
RANGE: [0, 100]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GSP_DONE
DESCRIPTION: Flag indicating completion of GSP task.
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [FALSE: running of task GSP incomplete, TRUE: running of task GSP complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: GV⁹³
DESCRIPTION: gain
USED IN: AECLP
UNITS: $\frac{\text{sec}}{\text{meter}}$
RANGE: [-5, 8]
DATA TYPE: **array (1..2) of real*8**
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GVE⁹⁴
DESCRIPTION: gain
USED IN: AECLP
UNITS: /second
RANGE: [0, 500]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GVEI⁹⁵
DESCRIPTION: gain
USED IN: AECLP
UNITS: /second²
RANGE: [-5, 40]
DATA TYPE: **array (1..2) of real*8**
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GVI⁹⁶
DESCRIPTION: gain
USED IN: AECLP
UNITS: /meter
RANGE: [-5, 5]
DATA TYPE: **array (1..2) of real*8**
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GW⁹⁷
DESCRIPTION: gain
USED IN: AECLP
UNITS: $\frac{\text{sec}}{\text{meter}}$
RANGE: [-5, 8]
DATA TYPE: **array (1..2) of real*8**
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GWI⁹⁸
DESCRIPTION: gain
USED IN: AECLP
UNITS: /meter
RANGE: [-5, 5]
DATA TYPE: **array (1..2) of real*8**
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: INIT_DONE
DESCRIPTION: Flag indicating completion of GCS initialization.
USED IN: 0. GCS
UNITS: none
RANGE: [FALSE: initialization incomplete, TRUE: initialization complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: INTERNAL_CMD⁹⁹
DESCRIPTION: Real vector containing the command to be sent to the axial engines
USED IN: AECLP
UNITS: none
RANGE: [-0.7, 1.7]
DATA TYPE: array (1..3) of real*8
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: TBD

NAME: K_ALT
DESCRIPTION: Determines use of altimeter radar by guidance processor
USED IN: ARSP, CP, GP
UNITS: none
RANGE: [0, 1]
DATA TYPE: array (0..4) of Integer*4
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: N/A

NAME: K_MATRIX
DESCRIPTION: Determines use of doppler radar by guidance processor.
USED IN: CP, GP, TDLRSP
UNITS: none
RANGE: [0, 1]
DATA TYPE: array (1..3, 1..3, 0..4) Integer*4
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: N/A

NAME: M1
DESCRIPTION: lower measured temperature calibration point for solid state temperature sensor
USED IN: TSP
UNITS: none
RANGE: [0, 2¹⁵-1]
DATA TYPE: Integer*2
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: M2
DESCRIPTION: upper measured temperature calibration point for solid state temperature sensor
USED IN: TSP
UNITS: none
RANGE: [0, 2¹⁵-1]
DATA TYPE: Integer*2
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: M3
DESCRIPTION: lower measured temperature calibration point for thermocouple pair temperature sensor
USED IN: TSP
UNITS: none
RANGE: [0, 2¹⁵-1]
DATA TYPE: Integer*2
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: M4
DESCRIPTION: upper measured temperature calibration point for thermocouple pair temperature sensor
USED IN: TSP
UNITS: none
RANGE: [0, 2¹⁵-1]
DATA TYPE: Integer*2
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: MAX_NORMAL_VELOCITY¹⁰⁰

DESCRIPTION: Maximum vertical velocity for safe landing

USED IN: GP

UNITS: $\frac{\text{meters}}{\text{sec}}$

RANGE: [0, 3.35]

DATA TYPE: real*8

ATTRIBUTE: data

DATA STORE LOCATION:

RUN_PARAMETERS

ACCURACY: N/A

NAME: OMEGA

DESCRIPTION: gain of angular velocity

USED IN: AECLP

UNITS: /second

RANGE: [-50, 50]

DATA TYPE: real*8

ATTRIBUTE: data

DATA STORE LOCATION: RUN_PARAMETERS

ACCURACY: N/A

NAME: P1

DESCRIPTION: pulse rate boundary

USED IN: RECLP

UNITS: $\frac{\text{radians}}{\text{sec}}$

RANGE: [0, 0.05]

DATA TYPE: real*8

ATTRIBUTE: data

DATA STORE LOCATION: RUN_PARAMETERS

ACCURACY: N/A

NAME: P2

DESCRIPTION: pulse rate boundary

USED IN: RECLP

UNITS: $\frac{\text{radians}}{\text{sec}}$

RANGE: [0, 0.05]

DATA TYPE: real*8

ATTRIBUTE: data

DATA STORE LOCATION: RUN_PARAMETERS

ACCURACY: N/A

NAME: P3

DESCRIPTION: pulse rate boundary

USED IN: RECLP

UNITS: $\frac{\text{radians}}{\text{sec}}$

RANGE: [0, 0.05]

DATA TYPE: real*8

ATTRIBUTE: data

DATA STORE LOCATION: RUN_PARAMETERS

ACCURACY: N/A

NAME: P4

DESCRIPTION: pulse rate boundary

USED IN: RECLP

UNITS: $\frac{\text{radians}}{\text{sec}}$

RANGE: [0, 0.05]

DATA TYPE: real*8

ATTRIBUTE: data

DATA STORE LOCATION: RUN_PARAMETERS

ACCURACY: N/A

NAME: PACKET

DESCRIPTION: Packet of telemetry data

USED IN: CP

UNITS: N/A

RANGE: N/A

DATA TYPE: array (1..256) of Integer*2

ATTRIBUTE: data

DATA STORE LOCATION: EXTERNAL

ACCURACY: N/A

NAME: PE_INTEGRAL¹⁰¹

DESCRIPTION: Integral portion of Pitch error equation

USED IN: AECLP, CP

UNITS: meters

RANGE: [-100, 100]

DATA TYPE: real*8

ATTRIBUTE: data

DATA STORE LOCATION: GUIDANCE_STATE

ACCURACY: TBD

NAME: PE_MAX

DESCRIPTION: Maximum pitch error tolerable

USED IN: AECLP

UNITS: none

RANGE: [0, 1]

DATA TYPE: **array(1..2) of** real*8

ATTRIBUTE: data

DATA STORE LOCATION: RUN_PARAMETERS

ACCURACY: N/A

NAME: PE_MIN

DESCRIPTION: Minimum pitch error tolerable.

USED IN: AECLP

UNITS: none

RANGE: [-1, 0]

DATA TYPE: **array(1..2) of** real*8

ATTRIBUTE: data

DATA STORE LOCATION: RUN_PARAMETERS

ACCURACY: N/A

NAME: RE_CMD¹⁰²
DESCRIPTION: roll engine command
USED IN: CP, RECLP
UNITS: none
RANGE: [1, 7]
D (direction) [0: positive, 1: negative]
I (intensity) [0: off, 1: minimum, 2: intermediate,
3: maximum]
DATA TYPE: Integer*2
ATTRIBUTE: data
DATA STORE LOCATION: EXTERNAL
ACCURACY: TBD

NAME: RE_STATUS
DESCRIPTION: status of the roll engines
USED IN: CP, RECLP
UNITS: none
RANGE: [0 : healthy, 1: failed]
DATA TYPE: logical*1
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: N/A

NAME: RE_SWITCH
DESCRIPTION: Flag indicating whether or not the
roll engines are turned on.
USED IN: GP, RECLP
UNITS: none
RANGE: [0: roll engines are off, 1: roll engines are
on.]
DATA TYPE: logical*1
ATTRIBUTE: data condition
DATA STORE LOCATION: GUIDANCE
ACCURACY: N/A

NAME: RECLP_DONE
DESCRIPTION: Flag indicating completion of
RECLP task.
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [FALSE: running of task RECLP
incomplete, TRUE: running of task RECLP
complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: RENDEZVOUS
DESCRIPTION: Control signal which indicates
whether or not GCS_SIM_RENDEZVOUS is to be
activated.
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [FALSE: GCS_SIM_RENDEZVOUS is
not to be activated, TRUE:
GCS_SIM_RENDEZVOUS is to be activated]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: RUN_DONE
DESCRIPTION: Flag indicating completion of GCS.
USED IN: 0. GCS
UNITS: none
RANGE: [FALSE: running of GCS incomplete,
TRUE: running of GCS complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: SP_DONE
DESCRIPTION: Control signal which indicates
whether or not Sensor Processing function has been
completed.
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [FALSE: running of Sensor Processing
function incomplete, TRUE: running of Sensor
Processing function complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: SS_TEMP
DESCRIPTION: Solid state temperature data
USED IN: TSP
UNITS: none
RANGE: [0, 2¹⁵-1]
DATA TYPE: Integer*2
ATTRIBUTE: data
DATA STORE LOCATION: EXTERNAL
ACCURACY: N/A

NAME: SUBFRAME_COUNTER
DESCRIPTION: Counter containing the number of the present subframe.
USED IN: CP
UNITS: none
RANGE: [1, 3]
DATA TYPE: Integer*2
ATTRIBUTE: data
DATA STORE LOCATION: EXTERNAL
ACCURACY: N/A

NAME: T1
DESCRIPTION: lower ambient temperature calibration point for solid state temperature sensor
USED IN: TSP
UNITS: degrees C
RANGE: [-250, 250]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: T2
DESCRIPTION: upper ambient temperature calibration point for solid state temperature sensor
USED IN: TSP
UNITS: degrees C
RANGE: [-250, 250]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: T3
DESCRIPTION: lower ambient temperature calibration point for thermocouple pair temperature sensor
USED IN: TSP
UNITS: degrees C
RANGE: [-50, 50]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: T4
DESCRIPTION: upper ambient temperature calibration point for thermocouple pair temperature sensor
USED IN: TSP
UNITS: degrees C
RANGE: [-50, 50]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: TD_COUNTER
DESCRIPTION: value returned by Touch Down Sensor
USED IN: TDSP
UNITS: none
RANGE: $[-2^{15}, 2^{15}-1]$
DATA TYPE: Integer*2
ATTRIBUTE: data
DATA STORE LOCATION: EXTERNAL
ACCURACY: N/A

NAME: TD_SENSED
DESCRIPTION: Flag indicating whether or not touch down has been sensed.
USED IN: CP, GP, TDSP
UNITS: none
RANGE: [0: touch down not sensed, 1: touch down sensed]
DATA TYPE: logical*1
ATTRIBUTE: data condition
DATA STORE LOCATION: SENSOR_OUTPUT
ACCURACY: N/A

NAME: TDLR_ANGLES¹⁰³
DESCRIPTION: vector of doppler radar beam offset angles (i.e., α , β , γ)
USED IN: TDLRSP
UNITS: radians
RANGE: $[0, \frac{\pi}{2})$
DATA TYPE: array (1..3) real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: TDLR_COUNTER
DESCRIPTION: value returned by Doppler radar
USED IN: TDLRSP
UNITS: none
RANGE: $[0, 2^{15}-1]$
DATA TYPE: array (1..4) Integer*2
ATTRIBUTE: data
DATA STORE LOCATION: EXTERNAL
ACCURACY: N/A

NAME: TDLR_GAIN¹⁰⁴
DESCRIPTION: gain in doppler radar beam
USED IN: TDLRSP

UNITS:

$\frac{\text{meters}}{\text{sec}}$

RANGE: [-1, 1]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS

ACCURACY: N/A

NAME: TDLR_LOCK_TIME
DESCRIPTION: locking time of doppler radar beam
USED IN: TDLRSP
UNITS: seconds
RANGE: [0, 60]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: TDLR_OFFSET
DESCRIPTION: offset in doppler radar beam
USED IN: TDLRSP
UNITS: $\frac{meters}{sec}$
RANGE: [-100, 0]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: TDLR_STATE¹⁰⁵
DESCRIPTION: state of the touch down landing radar beams.
USED IN: CP, TDLRSP
UNITS: none
RANGE: [0: Beam **unlocked**, 1: Beam **locked**]
DATA TYPE: array (1..4) logical*1
ATTRIBUTE: data condition
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: N/A

NAME: TDLR_STATUS
DESCRIPTION: status of the doppler radar
USED IN: CP, TDLRSP
UNITS: none
RANGE: [0 : healthy, 1: failed]
DATA TYPE: array (1..4) of logical*1
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: N/A

NAME: TDLR_VELOCITY
DESCRIPTION: Velocity as computed by the touch down landing radar.
USED IN: CP, GP, TDLRSP
UNITS: $\frac{meters}{sec}$
RANGE: [-100, 100]
DATA TYPE: array (1..3, 0..4) of real*8
ATTRIBUTE: data
DATA STORE LOCATION: SENSOR_OUTPUT
ACCURACY: TBD

NAME: TDLRSP_DONE
DESCRIPTION: Flag indicating completion of TDLRSP task.
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [FALSE: running of task TDLRSP incomplete, TRUE: running of task TDLRSP complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: TDSP_DONE
DESCRIPTION: Flag indicating completion of TDSP task.
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [FALSE: running of task TDSP incomplete, TRUE: running of task TDSP complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: TDS_STATUS
DESCRIPTION: status of the touch down sensor
USED IN: CP, GP, TDSP
UNITS: none
RANGE: [0 : healthy, 1: failed]
DATA TYPE: logical*1
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: N/A

NAME: TE_DROP
DESCRIPTION: The axial thrust error when axial engines are warm and the velocity altitude contour has not been intersected.
USED IN: AECLP
UNITS: none
RANGE: [-2, 2]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: TE_INIT
DESCRIPTION: The axial thrust error when the axial engines are cold.
USED IN: AECLP
UNITS: none
RANGE: [-2, 2]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: TE_INTEGRAL¹⁰⁶
DESCRIPTION: Integral portion of Thrust error equation
USED IN: AECLP, CP, GP
UNITS: meters
RANGE: [-100, 100]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: TBD

NAME: TE_LIMIT¹⁰⁷
DESCRIPTION: Limiting thrust error
USED IN: AECLP
UNITS: none
RANGE: [-100, 100]
DATA TYPE: real*8
ATTRIBUTE: Data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: TBD

NAME: TE_MAX
DESCRIPTION: Maximum thrust error **tolerable**
USED IN: AECLP
UNITS: none
RANGE: [-2, 2]
DATA TYPE: **array(1..2) of real*8**
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: TE_MIN
DESCRIPTION: Minimum thrust error tolerable.
USED IN: AECLP
UNITS: none
RANGE: [-2, 2]
DATA TYPE: **array(1..2) of real*8**
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: THERMO_TEMP
DESCRIPTION: thermocouple pair temperature
USED IN: TSP
UNITS: none
RANGE: [0, 2¹⁵-1]
DATA TYPE: Integer*2
ATTRIBUTE: data
DATA STORE LOCATION: EXTERNAL
ACCURACY: N/A

NAME: THETA¹⁰⁸
DESCRIPTION: **roll** angle
USED IN: RECLP
UNITS: radians
RANGE: [- π , π]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE
ACCURACY: TBD

NAME: THETA1
DESCRIPTION: pulse angle boundary
USED IN: RECLP
UNITS: radians
RANGE: [0, 0.05]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: THETA2
DESCRIPTION: pulse angle boundary
USED IN: RECLP
UNITS: radians
RANGE: [0, 0.05]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: TS_STATUS
DESCRIPTION: status of the temperature sensors in solid state, then thermocouple pair order
USED IN: CP, TSP
UNITS: none
RANGE: [0 : healthy, 1: failed]
DATA TYPE: array (1..2) of logical*1
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: N/A

NAME: TSP_DONE
DESCRIPTION: Flag indicating completion of TSP task.
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [FALSE: running of task TSP incomplete, TRUE: running of task TSP complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: VELOCITY_ERROR¹⁰⁹
DESCRIPTION: Distance from velocity-altitude contour. (Difference in velocities from actual to desired on contour.)
USED IN: AECLP, CP, GP
UNITS: $\frac{\text{meters}}{\text{sec}}$
RANGE: [-300, 20]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: TBD

NAME: YE_INTEGRAL¹¹⁰
DESCRIPTION: Integral portion of Yaw error equation
USED IN: AECLP, CP
UNITS: meters
RANGE: [-100, 100]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: TBD

NAME: YE_MAX
DESCRIPTION: Maximum yaw error **tolerable**
USED IN: AECLP
UNITS: none
RANGE: [-1, 1]
DATA TYPE: **array(1..2) of** real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: YE_MIN
DESCRIPTION: Minimum yaw error tolerable.
USED IN: AECLP
UNITS: none
RANGE: [-1, 1]
DATA TYPE: **array(1..2) of** real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

PART II. CONTENTS OF DATA STORES

Table 6.1: DATA STORE: GUIDANCE_STATE

VARIABLE NAME	USED BY:
A_STATUS	ASP, CP
AE_STATUS	AECLP, CP
AE_SWITCH	AECLP, GP
AE_TEMP	AECLP, CP, CRCP, GP
AR_STATUS	ARSP, CP
C_STATUS	CP
CHUTE_RELEASED	AECLP, CP, CRCP, GP
CL ¹¹¹	AECLP, GP
CONTOUR_CROSSED	AECLP, CP, GP
FRAME_BEAM_UNLOCKED	TDLRSP
FRAME_ENGINES_IGNITED	AECLP, GP
G_STATUS	CP, GSP
GP_ALTITUDE	CP, GP, AECLP
GP_ATTITUDE	AECLP, CP, GP
GP_PHASE	CP, GP
GP_ROTATION	AECLP, CP, GP
GP_VELOCITY	AECLP, CP, GP
INTERNAL_CMD	AECLP
K_ALT	ARSP, CP, GP
K_MATRIX	CP, GP, TDLRSP
PE_INTEGRAL	AECLP, CP
RE_STATUS	CP, RECLP
RE_SWITCH	GP, RECLP
TDLR_STATE	CP, TDLRSP
TDLR_STATUS	CP, TDLRSP
TDS_STATUS	CP, GP, TDSP
TE_INTEGRAL	AECLP, CP, GP
TE_LIMIT	AECLP
THETA	RECLP
TS_STATUS	CP, TSP
VELOCITY_ERROR	AECLP, CP, GP
YE_INTEGRAL	AECLP, CP

Table 6.2: DATA STORE: EXTERNAL

VARIABLE NAME	USED BY
A_COUNTER	ASP
AE_CMD	AECLP, CP
AR_COUNTER	ARSP
FRAME_COUNTER	AECLP, ARSP, CP, GP, TDLRSP
G_COUNTER	GSP
PACKET	CP
RE_CMD	RECLP, CP
SS_TEMP	TSP
SUBFRAME_COUNTER	CP
TD_COUNTER	TDSP
TDLR_COUNTER	TDLRSP
THERMO_TEMP	TSP

Table 6.3 : DATA STORE: SENSOR_OUTPUT

VARIABLE NAME	USED BY:
A_ACCELERATION	AECLP, ASP, CP, GP
AR_ALTITUDE	ARSP, CP, GP
ATMOSPHERIC_TEMP	ASP, CP, GSP, TSP
G_ROTATION	CP, GSP, GP, RECLP
TD_SENSED	CP, GP, TDSP
TDLR_VELOCITY	CP, GP, TDLRSP

Table 6.4: DATA STORE: RUN_PARAMETERS

VARIABLE NAME	USED BY
A_BIAS	ASP
A_GAIN_0	ASP
A_SCALE	ASP
ALPHA_MATRIX	ASP
AR_FREQUENCY	ARSP
COMM_SYNC_PATTERN	CP
CONTOUR_ALTITUDE	GP
CONTOUR_VELOCITY	GP
DELTA_T	AECLP, GP, RECLP, TDLRSP
DROP_HEIGHT	GP
DROP_SPEED ¹¹²	GP
ENGINES_ON_ALTITUDE	AECLP, GP
FULL_UP_TIME	AECLP
G1	ASP
G2	ASP
G3	GSP
G4	GSP
G_GAIN_0	GSP
G_OFFSET	GSP
GA	AECLP
GAX	AECLP
GP1	AECLP
GP2	AECLP
GPY	AECLP
GQ	AECLP
GR	AECLP
GRAVITY	AECLP, GP
GV	AECLP
GVE	AECLP
GVEI	AECLP
GVI	AECLP
GW	AECLP
GWI	AECLP
M1	TSP
M2	TSP
M3	TSP
M4	TSP
MAX_NORMAL_VELOCITY ¹¹³	GP
OMEGA	AECLP
P1	RECLP
P2	RECLP
P3	RECLP
P4	RECLP
PE_MAX	AECLP
PE_MIN	AECLP
T1	TSP
T2	TSP
T3	TSP
T4	TSP

Table 6.4 (continued): DATA STORE: RUN_PARAMETERS

VARIABLE NAME	USED BY
TDLR_ANGLES	TDLRSP
TDLR_GAIN	TDLRSP
TDLR_LOCK_TIME	TDLRSP
TDLR_OFFSET	TDLRSP
TE_DROP	AECLP
TE_INIT	AECLP
TE_MAX	AECLP
TE_MIN	AECLP
THETA1	RECLP
THETA2	RECLP
YE_MAX	AECLP
YE_MIN	AECLP

PART III. CONTROL SIGNALS, DATA CONDITIONS, AND GROUP FLOWS

Table 6.5: CONTROL SIGNALS (OPTIONAL USAGE)

CONTROL SIGNAL NAME
AECLP_DONE
ARSP_DONE
ASP_DONE
CLP_DONE
CP_DONE ¹¹⁴
CRCP_DONE
GP_DONE
GSP_DONE
INIT_DONE
RECLP_DONE ¹¹⁵
RENDEZVOUS
RUN_DONE ¹¹⁶
SP_DONE
TDLRSP_DONE
TDSP_DONE
TSP_DONE

Note: These variables are not in the required global data stores.

Table 6.6: DATA CONDITIONS (REQUIRED USAGE)

DATA CONDITION VARIABLE NAME
AE_SWITCH
AE_TEMP
CHUTE_RELEASED
CONTOUR_CROSSED
GP_PHASE
RE_SWITCH
TD_SENSED
TDLR_STATE

Table 6.7: INITIALIZATION DATA

VARIABLE NAME	USED BY
A_ACCELERATION	AECLP, ASP, CP, GP
A_BIAS	ASP
A_COUNTER	ASP
A_GAIN_0	ASP
A_SCALE	ASP
A_STATUS	ASP, CP
AE_STATUS	AECLP, CP
AE_SWITCH	AECLP, GP
AE_TEMP	AECLP, CP, CRCP, GP
ALPHA_MATRIX	ASP
AR_ALTITUDE	ARSP, CP, GP
AR_COUNTER	ARSP
AR_FREQUENCY	ARSP
AR_STATUS	ARSP, CP
ATMOSPHERIC_TEMP	ASP, CP, GSP, TSP
C_STATUS	CP
CHUTE_RELEASED	AECLP, CP, CRCP, GP
CL ¹¹⁷	AECLP, GP
COMM_SYNC_PATTERN	CP
CONTOUR_ALTITUDE	GP
CONTOUR_CROSSED	AECLP, CP, GP
CONTOUR_VELOCITY	GP
DELTA_T	AECLP, GP, RECLP, TDLRSP
DROP_HEIGHT	GP
DROP_SPEED ¹¹⁸	GP
ENGINES_ON_ALTITUDE	AECLP, GP
FRAME_BEAM_UNLOCKED	TDLRSP
FRAME_COUNTER	AECLP, ARSP, CP, GP, TDLRSP
FRAME_ENGINES_IGNITED	AECLP, GP
FULL_UP_TIME	AECLP
G1	ASP
G2	ASP
G3	GSP
G4	GSP
G_COUNTER	GSP
G_GAIN_0	GSP
G_OFFSET	GSP
G_ROTATION	CP, GSP, GP, RECLP
G_STATUS	CP, GSP
GA	AECLP
GAX	AECLP
GP1	AECLP
GP2	AECLP
GP_ALTITUDE	AECLP, CP, GP
GP_ATTITUDE	AECLP, CP, GP
GP_PHASE	CP, GP
GP_ROTATION	AECLP, CP, GP
GP_VELOCITY	AECLP, CP, GP
GPY	AECLP
GQ	AECLP
GR	AECLP
GRAVITY	AECLP, GP
GV	AECLP

Table 6.7 (continued): INITIALIZATION DATA

VARIABLE NAME	USED BY
GVE	AECLP
GVEI	AECLP
GVI	AECLP
GW	AECLP
GW1	AECLP
K_ALT	ARSP, CP, GP
K_MATRIX	CP, GP, TDLRSP
M1	TSP
M2	TSP
M3	TSP
M4	TSP
MAX_NORMAL_VELOCITY ¹¹⁹	GP
OMEGA	AECLP
P1	RECLP
P2	RECLP
P3	RECLP
P4	RECLP
PE_INTEGRAL	AECLP, CP
PE_MAX	AECLP
PE_MIN	AECLP
RE_STATUS	CP, RECLP
RE_SWITCH	GP, RECLP
SS_TEMP	TSP
SUBFRAME_COUNTER	CP
T1	TSP
T2	TSP
T3	TSP
T4	TSP
TD_COUNTER	TDSP
TD_SENSED	CP, GP, TDSP
TDLR_ANGLES	TDLRSP
TDLR_COUNTER	TDLRSP
TDLR_GAIN	TDLRSP
TDLR_LOCK_TIME	TDLRSP
TDLR_OFFSET	TDLRSP
TDLR_STATE	CP, TDLRSP
TDLR_STATUS	CP, TDLRSP
TDLR_VELOCITY	CP, GP, TDLRSP
TDS_STATUS	CP, GP, TDSP
TE_DROP	AECLP
TE_INIT	AECLP
TE_INTEGRAL	AECLP, CP, GP
TE_LIMIT	AECLP
TE_MAX	AECLP
TE_MIN	AECLP
THERMO_TEMP	TSP
THETA	RECLP
THETA1	RECLP
THETA2	RECLP
TS_STATUS	CP, TSP
VELOCITY_ERROR	AECLP, CP, GP
YE_INTEGRAL	AECLP, CP
YE_MAX	AECLP

Table 6.8: TEMP_DATA

VARIABLE NAME
SS_TEMP
THERMO_TEMP

Table 6.9: SENSOR_DATA

VARIABLE NAME
A_COUNTER
AR_COUNTER
TDLR_COUNTER
G_COUNTER
TEMP_DATA
TD_COUNTER

Table 6.10: OUTPUT_DATA

VARIABLE NAME
AE_CMD
RE_CMD
PACKET

Table 6.11: OUTPUT_CONTROL

VARIABLE NAME
AE_SWITCH
RE_SWITCH
CHUTE_RELEASED

Table 6.12: FRAME_DATA

VARIABLE NAME

FRAME_COUNTER
SUBFRAME_COUNTER

A. NOTATION FOR LEVELS 0, 1, 2, AND 3 SPECIFICATION

A. NOTATION FOR LEVELS 0, 1, 2, AND 3 SPECIFICATION

This specification **was developed using** the extended structured analysis method advocated by Hatley [12, 13] and Cadre's *teamwork* [19]. This method is based on a hierarchical approach to defining functional modules and the associated data and control flows.

The documents constructed as a part of this specification include data context and flow diagrams; control context and flow diagrams; process and control descriptions; and a Data Requirements Dictionary. Figure A.1 defines the graphical symbols used in the data flow and control flow diagrams, respectively.

The data flow diagrams describe the processes, data flows, and data stores. The data context diagram is the highest-level data flow diagram and represents the data flow for the entire system.

The control flow diagrams describe processes, control signal and data condition flows, control specifications, and data stores. The control signal and data condition flows are depicted using directed arcs with broken lines. The control signals listed in the data dictionary may be implemented by the programmer in any form desired; or, they may be completely ignored and the control of the program conducted through other means. The **control signals** simply show the logic involved in the system. Signal flows between the control flow diagram and the control specification have a short bar at the end of the directed arc. The control flow diagrams contain duplicate descriptions of the processes represented on the data flow diagram. The control context diagram representing the most abstract control flow is similar to the data context diagram.

The control specifications describe the control requirements of a system. These specifications contain the conditions when the processes detailed in the data and control flow diagrams are activated and de-activated.

The Data Requirements Dictionary contains definitions for data, data conditions, control signals, and group flows.

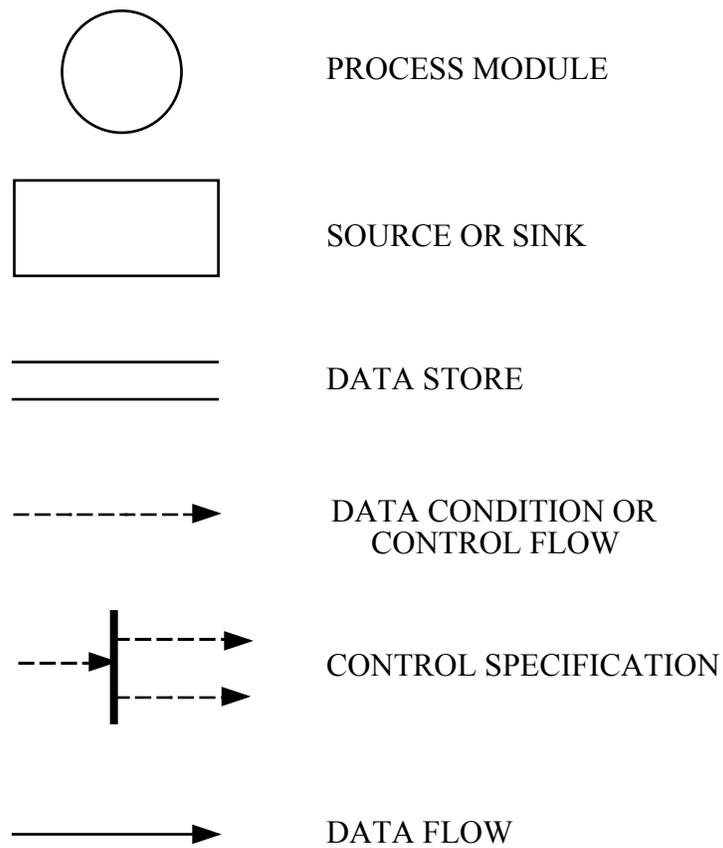
Following is a list of definitions and explanations for the structured analysis diagrams:

1. The data and control flow names on the directed arcs in the structured analysis figures can be found in the Data Requirements Dictionary Part I, while the group flow names on the arcs can be found in the Data Requirements Dictionary Part III.
2. In the Process Activation Tables, the first column contains the inputs. The second set of columns (separated by two vertical lines) contains the cells which indicate whether a process is to be activated or deactivated. A blank cell indicates that the process is deactivated. An integer indicates that the process is activated. A process whose cell contains the integer "n" must complete before the process with integer "n+1" is activated. All processes whose cells contain the same integer can be activated in any order. The third set of columns, if present, represents the output values for control signals.
3. The meanings for the symbols used in the expressions for inputs are:

=	equal
~=	not equal
~	logical NOT

- & logical AND
- | logical OR
- () grouping (expression inside parentheses is evaluated first)

Figure A.1: GRAPHICAL SYMBOLS USED IN STRUCTURED ANALYSIS DIAGRAMS



B. IMPLEMENTATION NOTES

INTERFACE

Background

For the purposes of this research experiment, each GCS implementation must function as if it were actually controlling a planetary lander. In reality, each GCS implementation will be interacting with a software simulator (GCS_SIM) that *models* the behavior of a physical lander when exposed to the environmental forces of a planet.

Due to the fact that each GCS implementation must interact with GCS_SIM as if it were connected to the lander hardware, there are some additional requirements that are placed on a GCS implementation that help define a *software* interface. The software interface to the simulator replaces the physical connection to planetary lander hardware through the use of a simulator support utility and an additional requirement involving the organization of the global data stores.

Simulator Support Utility

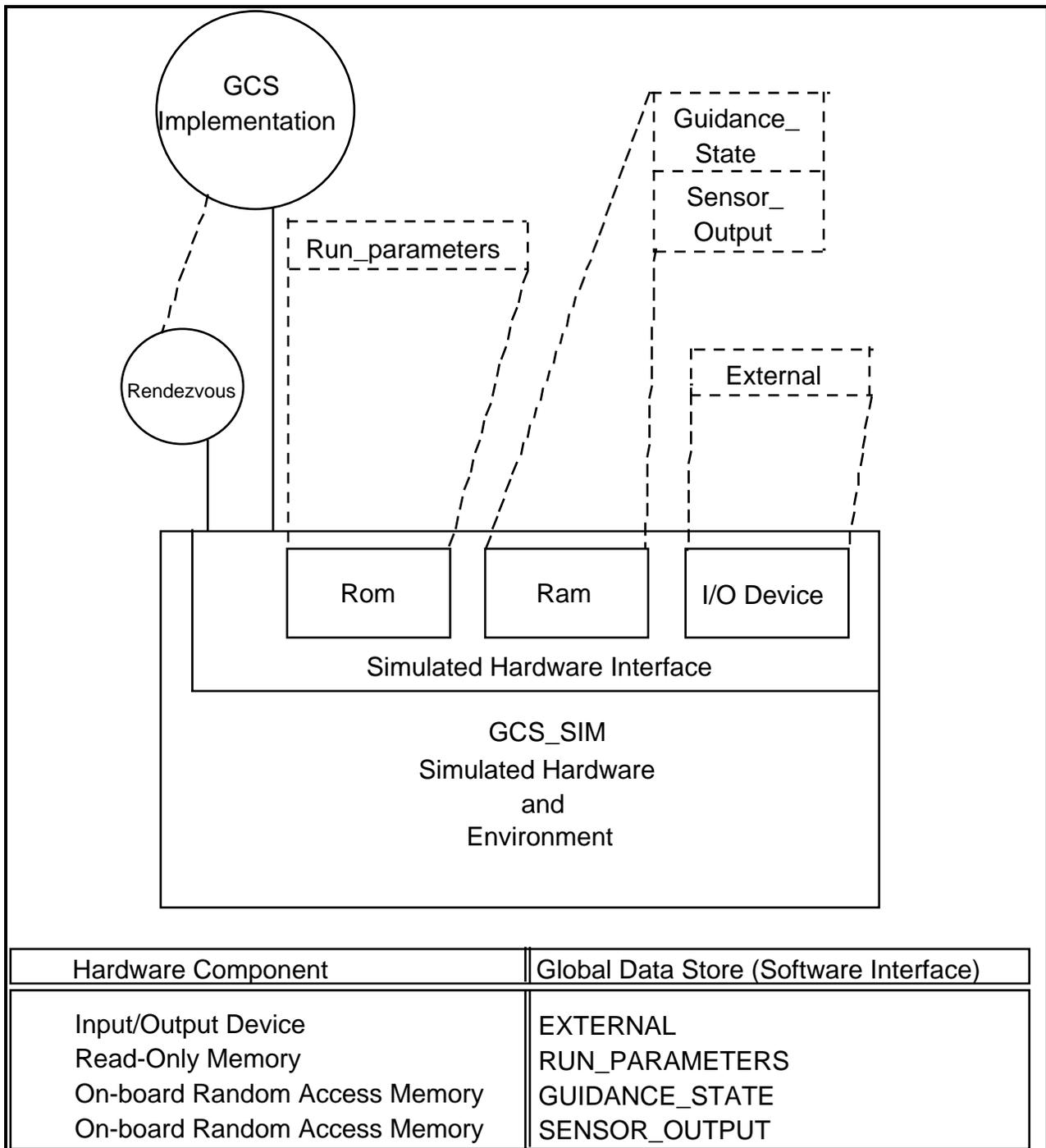
A single simulator support utility (GCS_SIM_RENDEZVOUS) is provided to form a uniform interface between the GCS **implementation** and the simulation environment (GCS_SIM). This utility is a routine which simplifies the interface between the GCS implementations and the simulation of the vehicle sensing and control mechanisms. This utility also includes a synchronization mechanism for the configurations using more than one version of the GCS. This routine provides the following support functions:

- Initialization for the Beginning of Terminal Descent
- Simulator Rendezvous Synchronization
- GCS Interface for Simulated Reads and Writes

Input/Output

The GCS_SIM_RENDEZVOUS routine simulates all of the input/output operations for each GCS implementation. When using the rendezvous routine with a GCS implementation, all data needed by rendezvous is passed via the four global data stores and there are no additional parameters required. All information *read from* or *written to* each GCS **implementation** will be transferred through the four global data stores defined in the data dictionary.

Figure B.1: DIAGRAM OF STORAGE AS SEEN BY GCS IMPLEMENTATIONS



Process

The GCS **uses** the sensor input values **in order to calculate** control commands which **are used by GCS_SIM to manipulate the actuators**. Since GCS_SIM handles the *orbit to terminal descent* portion of each trajectory, a rendezvous must be issued at the start of each trajectory to load initial sensor values into each GCS **implementation**. Following the first call to rendezvous, all GCS **implementations** will synchronize themselves by calling rendezvous **prior to the execution** of each subframe. This rendezvous, in effect, suspends the GCS implementations until the other GCS implementations have processed this time step or have run out of time.¹²⁰

The calling convention for this GCS_SIM provided support utility is as follows:

- GCS_SIM_RENDEZVOUS (*requires no parameters*)

GCS Initialization

During the initialization phase of each GCS trajectory (the first call to **GCS_SIM_RENDEZVOUS**) the frame counter (**FRAME_COUNTER**) will be updated with the starting frame **number** for the particular trajectory. Under *normal* circumstances, the value of the frame counter will be "1," but **the programmer should not** rely on that.¹²¹

By using the interface described above, the simulator can be transparent to the implementation.

C. NUMERICAL INTEGRATION INSTRUCTIONS

NUMERICAL INTEGRATION INSTRUCTIONS

Within the Guidance Processing functional unit, the calculations of GP_VELOCITY, GP_ALTITUDE, and GP_ATTITUDE require the use of a highly accurate integration method. To maintain the necessary degree of accuracy, three methods of numerical integration have been designated as acceptable for coding, namely Adams-Moulton method, Hamming's method, and the Runge-Kutta fourth-order method **for simultaneous equations**. If the Runge-Kutta method is used, it is required that the three equations be solved as a set of simultaneous equations.¹²²

Each method is briefly described in the following paragraphs, and references to numerical analysis texts describing the method are provided. Algorithms specified in either a text listed or another suitable numerical analysis text should be used during coding.

Adams-Moulton Method

The Adams-Moulton Method requires values from the previous four time steps to calculate the value at the next time step. The Adams-Moulton method is a predictor/corrector method. Both [14] (pp. 346-7) and [16] (pp. 478-81) explain the Adams-Moulton method.

Hamming's Method

The Hamming method uses a predictor/corrector method similar to that of Adams-Moulton. Hamming's method uses the same predictor as Milne's, but uses a much simpler corrector formula. Milne's method of integration was deemed too unstable for use, but Hamming's method with the simpler corrector is sufficiently stable. A description of both Hamming's method and Milne's method can be found in [14] (pp. 347-8).

Runge-Kutta Fourth-Order Method for Simultaneous Equations¹²³

The well-known Runge-Kutta fourth-order method **for simultaneous equations** requires only the previous two values to calculate the next value. References can be found in many texts including: [15](pp. 356-60), [17] (pp. 240-6; pp. 282-5), [18] (pg. 447; pp. 471-3)

During the first time step, using a numerical integration method necessitates some specification of previous values. These values will be provided during initialization for the data elements provided in Table C.1.

TABLE C.1: INITIAL VALUES PROVIDED FOR USE IN INTEGRATION

A_ACCELERATION (1..3, 0..4)
AR_ALTITUDE (0..4)
GP_ALTITUDE (0..4)
GP_ATTITUDE (1..3, 1..3, 0..4)
GP_VELOCITY (1..3, 0..4)
G_ROTATION (1..3, 0..4)
K_ALT (0..4)
K_MATRIX (1..3, 1..3, 0..4)
TDLR_VELOCITY (1..3, 0..4)

Note that not all integration required by the GCS specification requires the use of one of the methods listed in this appendix. More specifically, in computing **THETA**, TE_INTEGRAL, PE_INTEGRAL, and YE_INTEGRAL, Euler's method provides sufficient accuracy and simplicity and should be used. Information on Euler's method may be found in: [14](pp. 318-22), [15](pg. 223), and [16](pp. 462-3).

BIBLIOGRAPHY

- [1] Federal Aviation Administration. One McPherson Square, 1425 K Street N.W., Suite 500 Washington, DC 20005, Radio Technical Commission for Aeronautics Document RTCA/DO-178A, August 1986.
- [2] George B. Finelli. Results of software error-data experiments. In AIAA/AHS/ASEE Aircraft Design, Systems and Operations Conference, Atlanta, GA, September 1988.
- [3] Harm Buning and D. T. Greenwood. Flight mechanics for space and re-entry vehicles. Technical report, The University of Michigan Engineering Summer Conferences, Summer 1964.
- [4] Herbert Goldstein. Classical Mechanics. Addison-Wesley Publishing Company, Inc., Reading, Massachusetts, USA, 1959.
- [5] Irving H. Shames. Engineering Mechanics -- Statics and Dynamics. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1980.
- [6] Dan Edwin Christie. Vector Mechanics. McGraw-Hill Inc., New York, 1964
- [7] David Hestenes. New Foundations for Classical Mechanics. D. Reidel Publishing Company, Boston, 1986
- [8] D. N. Burghes and A. M. Downs. Classical Mechanics and Control. Ellis Horwood Limited, Coll House, Westergate, England, 1975.
- [9] G. S. Light and J. B. Higham. Theoretical Mechanics. Longman Inc., New York, 1975.
- [10] Don C. Rich and J. R. Dunham. Guidance and Control Software Simulator (GCS_SIM) Software Specification. Technical Report NASA Contract NAS1-17964; Task Assignment No. 8, Research Triangle Institute, Research Triangle Park, NC, 1987.
- [11] Andrew S. Tanenbaum. Computer Networks. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1981.
- [12] Derek J. Hatley. The use of structured methods in the development of large, software-based avionics systems. In Proceedings of the AIAA/IEEE 6th Digital Avionics Systems Conference, New York, December 1984.
- [13] Derek J. Hatley and Imtiaz A. Pirbhai, Strategies for Real-Time System Specification. Dorset House Publishing Company, New York, New York, 1987.
- [14] W. Allen Smith. Elementary Numerical Analysis. Harper and Row, New York, 1979.
- [15] J. B. Scarborough. Numerical Mathematical Analysis. The Johns Hopkins Press, Baltimore, 1962
- [16] Stephen M. Pizer. Numerical Computing and Mathematical Analysis. Science Research Associates, Inc., Chicago, 1975
- [17] **Richard L. Burden and J. Douglas Faires. Numerical Analysis. PWS-KENT Publishing Company, Boston, 1989.**
- [18] **Melvin J. Maron and Robert J. Lopez. Numerical Analysis: A Practical Approach. Wadsworth Publishing Company, Belmont, California, 1990.**
- [19] teamwork/SA teamwork/RT User's Guide, Cadre Technologies, Inc., Providence, Rhode Island, Release 4.0, 1990.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33

34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68

69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103

104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123