NASA Contractor Report 182058

SOFTWARE REQUIREMENTS

Guidance and Control Software Development Specification

B. Edward Withers, Don C. Rich, Douglas S. Lowman, and R. C. Buckland

RESEARCH TRIANGLE INSTITUTE
Research Triangle Park, North Carolina

Contract NAS1-17964
June 1990

National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665-5225

SOFTWARE REQUIREMENTS

Guidance and Control Software

Development Specification

RTCA DO-178A Document Number 2

Release number: 2.1

Prepared for:

NASA-Langley Research Center under contract
NAS1-17964; Task Assignment No. 8.

Prepared by:

Author(s): B. Edward Withers
Don C. Rich
Douglas S. Lowman
R. C. Buckland
Reviewer(s): RTI - Anita M. Shagnea
Janet R. Dunham
NASA - G. Earle Migneault
Bernice Becher
George B. Finelli

Software R & D Department
Center for Digital Systems Research
Research Triangle Institute
Research Triangle Park, North Carolina 27709

ACKNOWLEDGEMENT

These specifications were reverse-engineered from a simulation program written by Earle Migneault during the early seventies to study the probability of success of the 1976 Viking Lander missions to Mars. We are grateful to him for identifying such an interesting problem to be studied.

Preface

The Guidance and Control Software Development Specification is document # 2 in a series of fifteen documents which fulfill the Radio Technical Commission for Aeronautics RTCA/DO-178A guidelines, "Software Considerations in Airborne Systems and Equipment Certification [1]." The documents are numbered as specified in the DO-178A guidelines. The documents in the series are used to demonstrate compliance with the DO-178A guidelines by describing the application of the procedures and techniques used during the development of flight software. These documents were prepared under contract with NASA-Langley Research Center as a part of their long term research program addressing the fundamentals of the software failure process.

This project consists of two complementary goals: first, to develop software for use by the Research Triangle Institute (RTI) in the software error studies research program sponsored by NASA-Langley Research Center [2]; second, to use and assess the RTCA/DO-178A guidelines for the Federal Aviation Administration (FAA). The two goals are complementary in that the use of the structured DO-178A guidelines in the development of the software will ensure that the test specimens of software have been developed according to the industry standards for flight critical software. The error studies research analyses will then be conducted using high quality software specimens.

The implementations will be subjected to two different software testing environments: verification of each implementation according to the RTCA/DO-178A guidelines and replicated random testing in a configuration which runs more than one test specimen at a time. The term *implementations* refers to bodies of code written by different programmers, while a *version* is a piece of code at a particular state (i.e., version 2.0 is the result of code review). This research effort involves the gathering of product and process data from every phase of software development for later analysis. More information on the goals of the Guidance and Control Software (GCS) project are available in the *GCS Plan for Software Aspects of Certification.*

The series consists of the following documents:

- *GCS Configuration Index* Document no. 1
- *GCS Development Specification* Document no. 2

- *GCS Design Descriptions* One for each software implementation. Document no. 3
- *GCS Programmer's Manual* Document no. 4, 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* One for each software implementation. Document no. 6
- *GCS Source Code* One for each software implementation. Document no. 7
- *GCS Executable Object Code* One for each software implementation. Not available on hardcopy. Document no. 8
- *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

FOREWORD

This specification defines the fourth problem to be studied as a part of a series of controlled case studies sponsored by NASA-Langley Research Center. These studies address the fundamentals of the software failure process. The goal is to develop a method for assessing, and engineering, reliable and safe software.

This fourth problem, a guidance and control system for a planetary landing vehicle, represents an order of magnitude increase in problem complexity over the previous problems studied. It is specified using an extension to the popular method of structured analysis. This specification method was selected instead of a formal one for the sole purpose of not making the specification development activity a research effort in itself. In addition, the intent of the study is to observe failures, given that the software has been developed using a quality-oriented, state-of-the-art engineering approach.

Note that 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. In addition, an individual well-versed in aeronautical engineering should be available to answer programming questions concerning vehicle dynamics.

Much effort has been expended in making this specification as error free as possible. It has been validated by extensive peer review and informal walkthrough, coding a prototype implementation, and using an extended structured analysis design tool.

Janet R. Dunham

Edward Withers

March 18, 1988

Contents

List of Figures

List of Tables

PAGE XIV INTENTIONALLY BLANK

1. INTRODUCTION

PURPOSE OF THE GUIDANCE AND CONTROL SOFTWARE

The purpose of the Guidance and Control Software (GCS) 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 terminal phase of descent trajectory is shown in Figure 1.2.

The initialization of the GCS starts the sensing of vehicle altitude. When a pre-defined 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 attempts to maintain the descent of the vehicle along a pre-determined velocity-altitude contour. The vehicle descends along this contour until a pre-defined 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 DESCENT

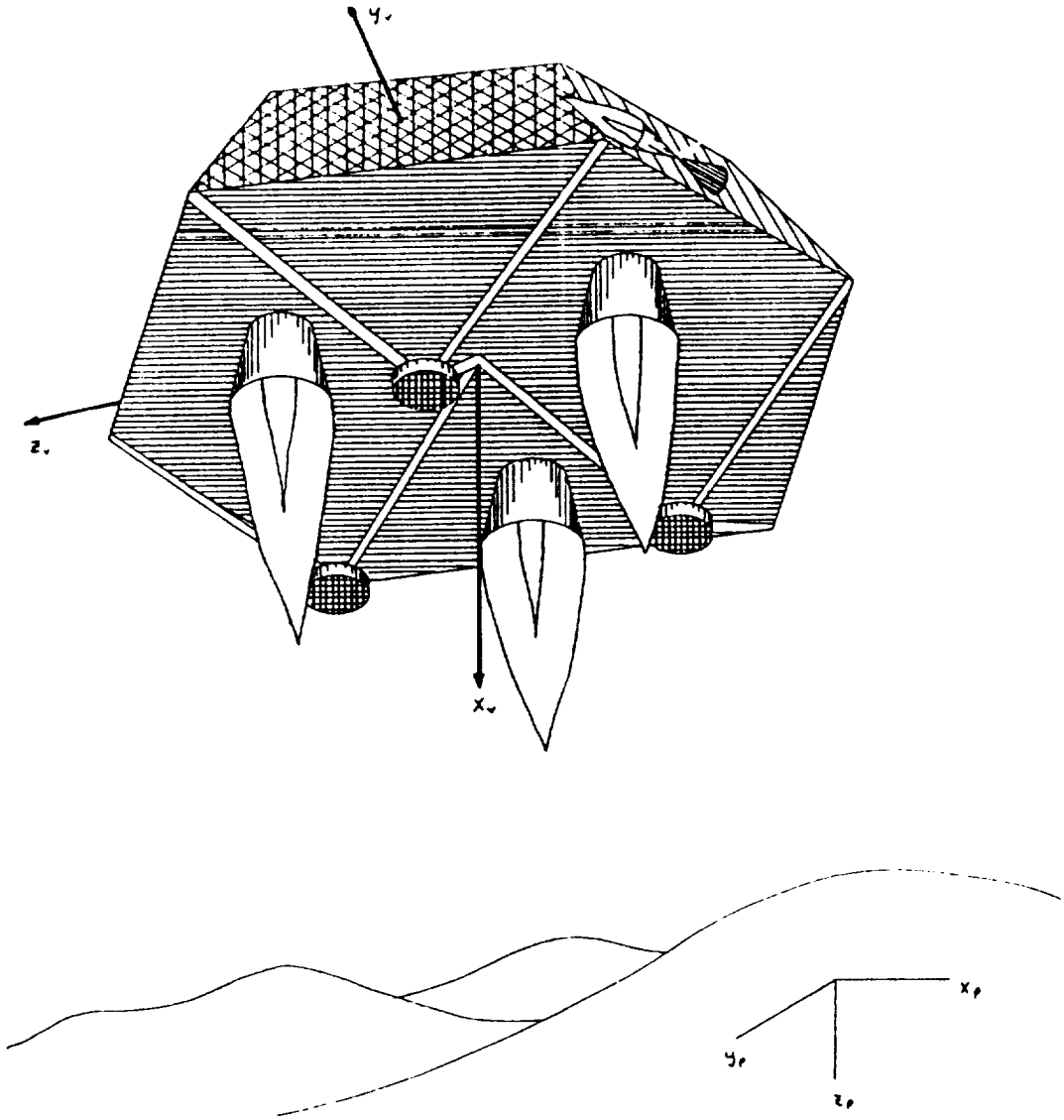
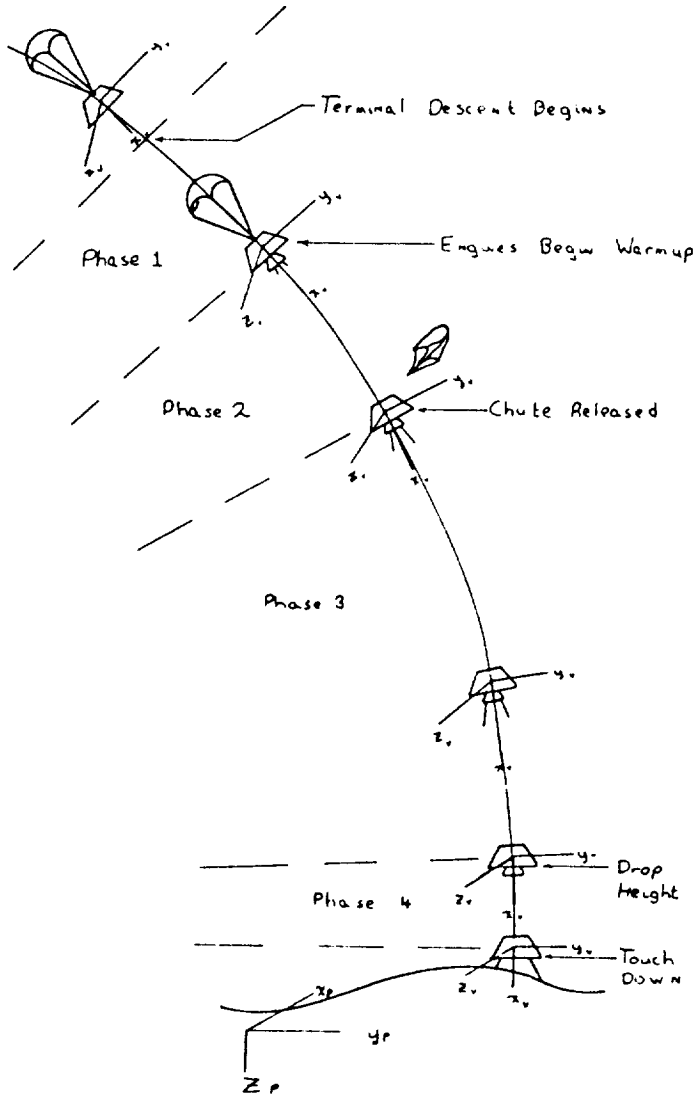


Figure 1.2: A TYPICAL TERMINAL DESCENT TRAJECTORY



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 pre-determined altitude is reached. The atmosphere introduces drag forces, including the random effects of wind. Differentially 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 pre-determined velocity altitude contour. At approximately 60 feet above the planet surface, the vehicle is maintained at a constant descent velocity of ten feet per second. 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,

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

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 (˙) 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

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

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].

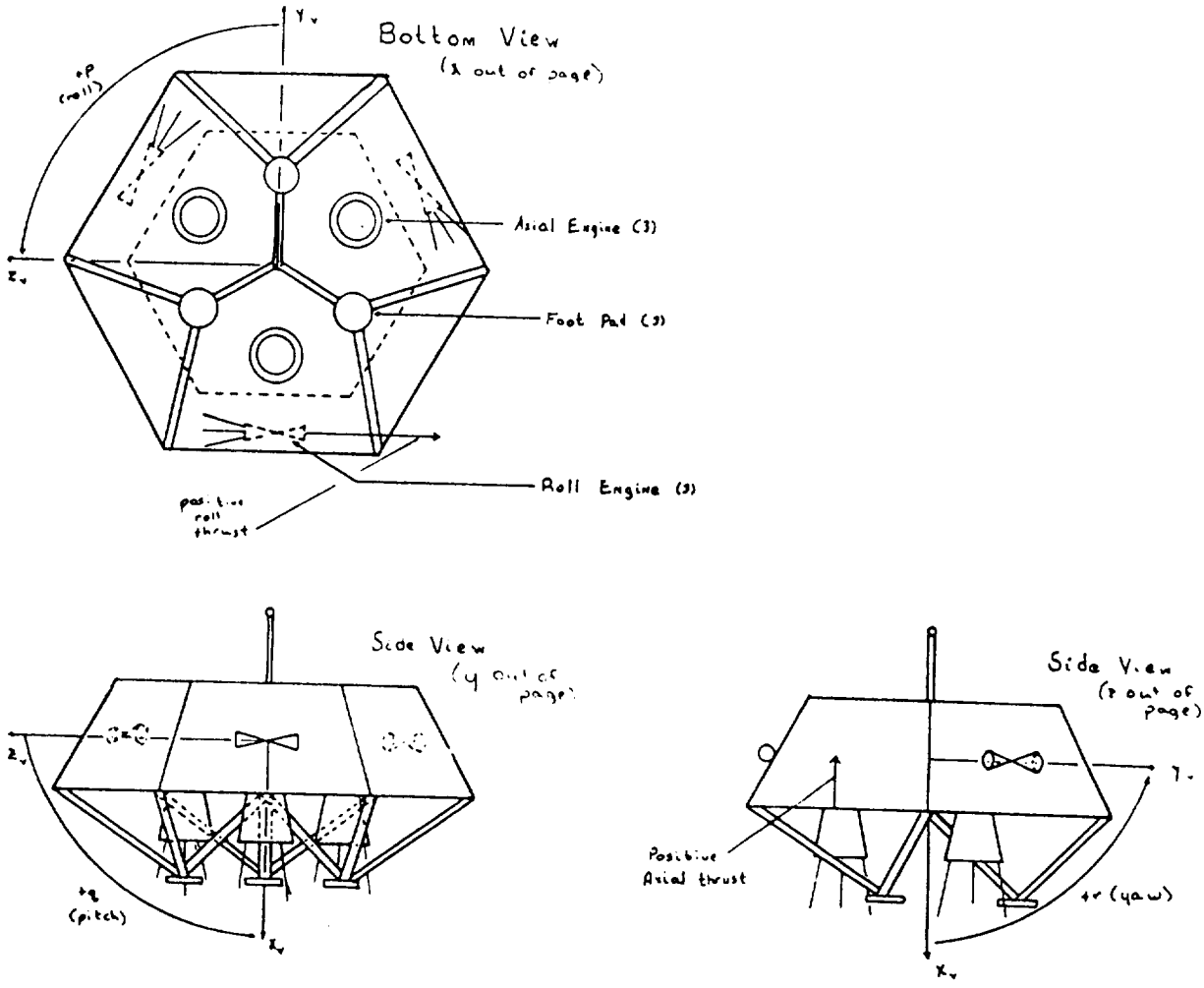
Vehicle Attitude

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 $\ddot{x}_v$, $\ddot{y}_v$, and $\ddot{z}_v$ respectively.

Figure 1.3: ENGINEERING ILLUSTRATION OF VEHICLE



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.

Notation

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 ($\cdot$) Small dots are used to denote scalar multiplication. For example:

$$3 \cdot 4 = 12$$

Multiplication sign $\times$ 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 then becomes:

$$A\_ACCELERATION\_M(i) = A\_BIAS(i) + A\_GAIN(i) * A\_COUNTER(i)$$

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:

$$A\_ACCELERATION\_M(1) = A\_BIAS(1) + A\_GAIN(1) * A\_COUNTER(1).$$

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.

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.

VEHICLE GUIDANCE

Vehicle guidance is accomplished by varying the engine thrust so that the vehicle follows a single pre-determined 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.

2. LEVEL 0 SPECIFICATION

The GCS will operate within a redundant, distributed-processing framework. It 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 pre-determined velocity-altitude contour which has been chosen to conserve enough fuel to effect a safe attitude and impact upon landing.

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 pre-determined velocity/altitude contour.

The GCS also transmits telemetry data and rendezvous with GCS_SIM [10], the simulator and controller.

Versions of the GCS developed from this specification may be executed singly or in parallel. Output from multiple versions at various synchronization points will be voted to control the vehicle. One of the effects of this design on the specification has been a constraint to use only specific system services. In particular, a rendezvous routine will be provided and should be invoked, as specified in the implementation_notes [Appendix B]. Other system services and library routines are explicitly excluded from use by the programmers.

When programming, the modules shown in this specification need not be treated as totally separate units. The programmer determines the organization of the code with two mandatory requirements. The data stores must be organized as given, and the code must work within the context of the

timing requirements of the system as given. For purposes of flight system design, all components of the system are considered flight critical as defined by RTCA document DO-178A[1].

Figure 2.1: DATA CONTEXT DIAGRAM FOR THE GCS

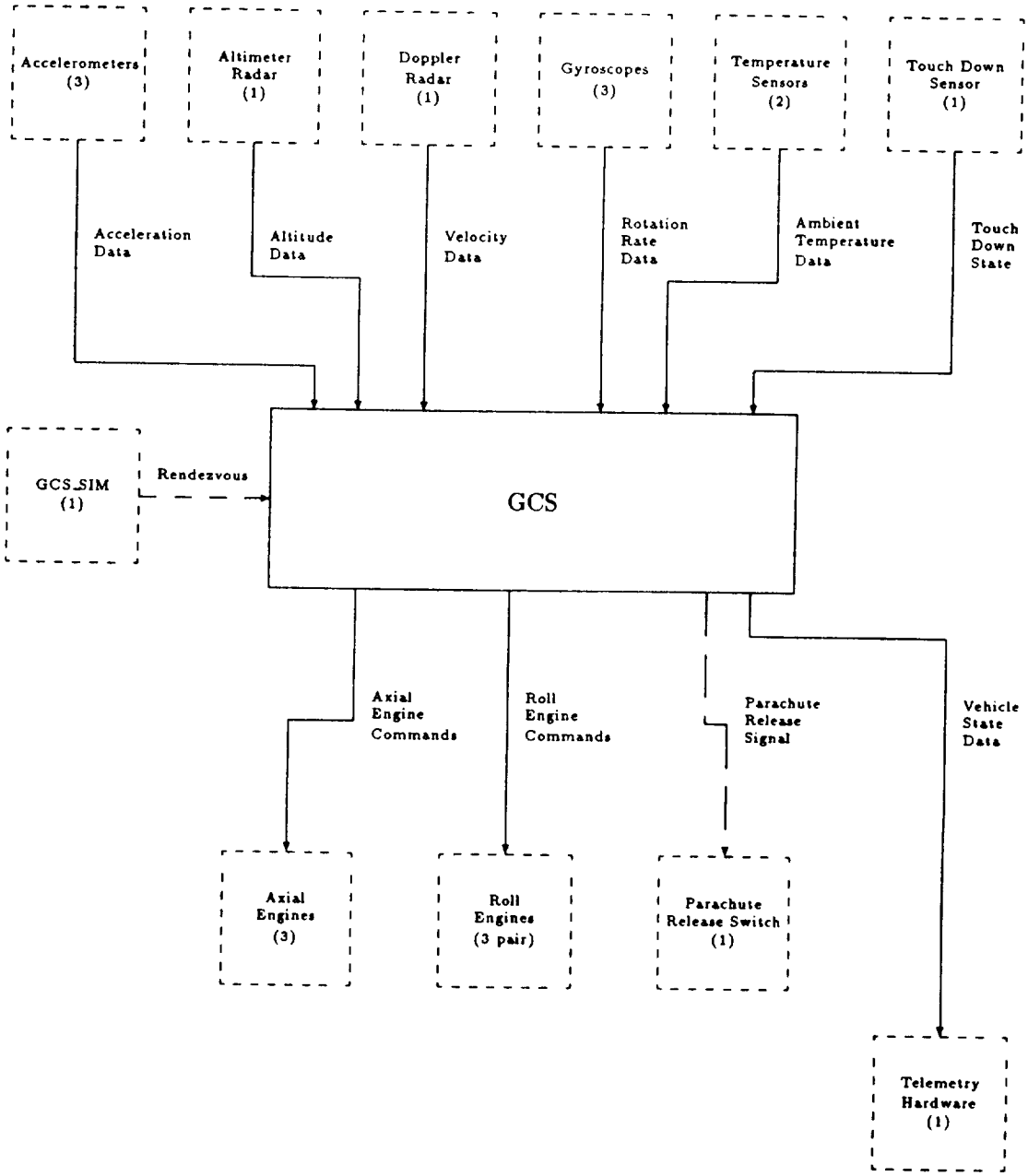
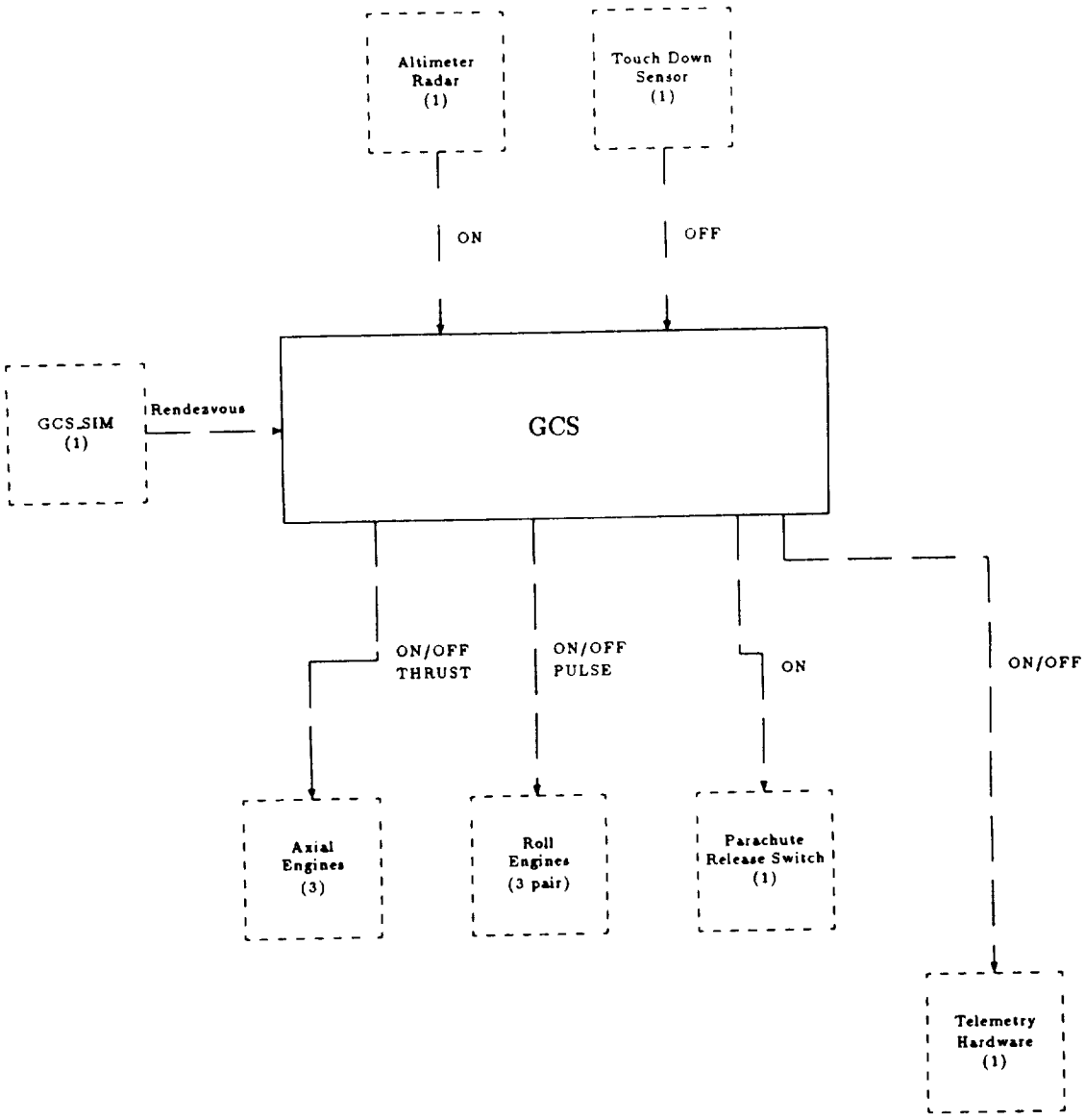


Figure 2.2: GCS CONTROL CONTEXT DIAGRAM



3. LEVEL 1 SPECIFICATION

Figure 3.1: PROCESS 0. GCS

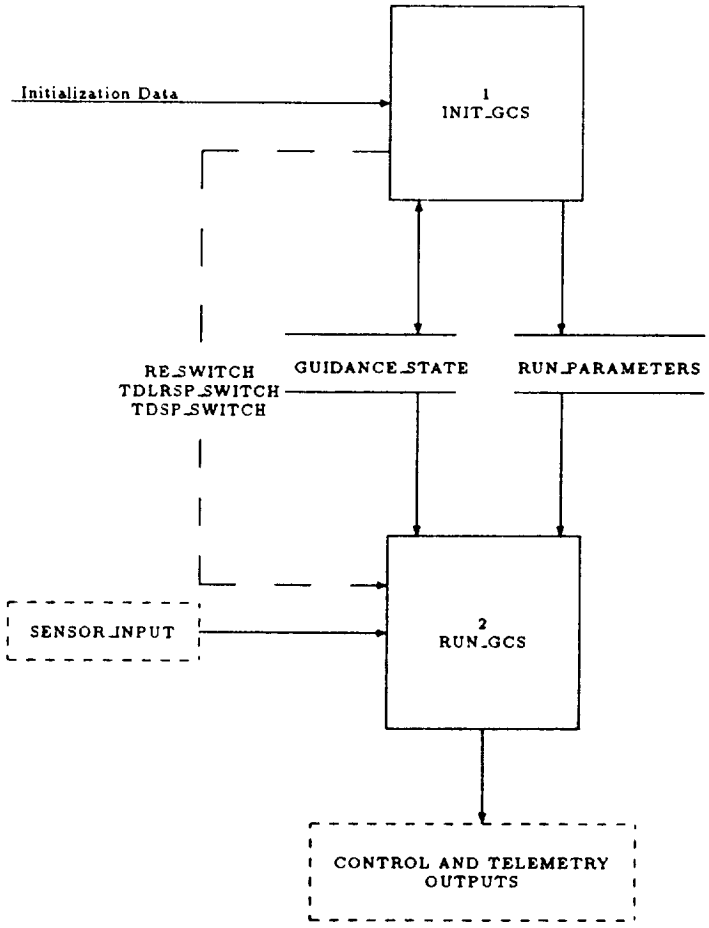


Figure 3.2: CONTROL 0. GCS

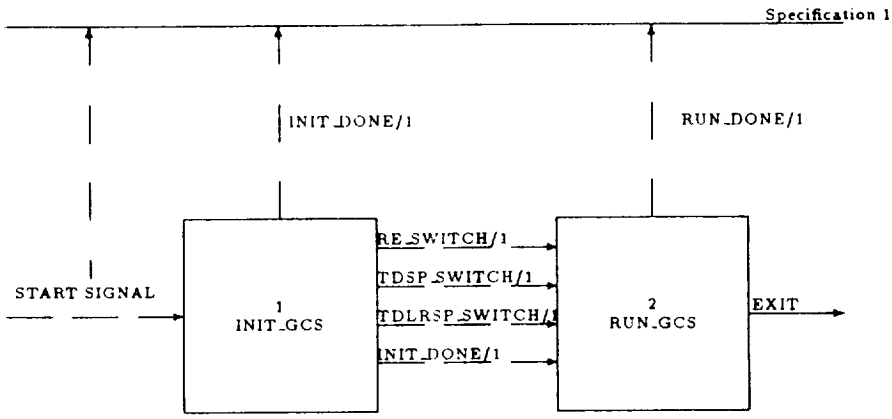


Table 3.1: CONTROL 0. GCS - SPECIFICATION 1

CONTROL VARIABLE	STATE
Start Signal = 1	Init_GCS
INIT_DONE = 1	Run_GCS
RUN_DONE = 1	Inactive

4. LEVEL 2 SPECIFICATION

PROCESS 1. INIT_GCS

PURPOSE INIT_GCS initializes the guidance and control software.

INPUT None

OUTPUT See Tables 7.8-7.11.

PROCESS Init_GCS will be executed on the first call to rendezvous. Both Init_GCS and rendezvous will be supplied to the programmer. There should be a call to rendezvous prior to executing each sub-frame. The first call will execute Init_GCS, which will load any needed initial values for later use.

- LOAD INITIAL VALUES - Load initial values for velocity, altitude, and attitude, as well as any others, such as the constant gains and offsets that are needed. The values to be loaded in are shown in the table INITIALIZATION DATA in part III of the DATA DICTIONARY.
- TURN ON SWITCHES - Turn on the Roll Engine Switch, the Touch Down Landing Radar Switch, and the Touch Down Sensor Switch.
- SET FRAME COUNTER - The frame counter will be initialized to some number representing the next frame to be executed. This allows the option of starting execution at some point beyond the first frame of a trajectory.

Figure 4.1: PROCESS 2. RUN_GCS

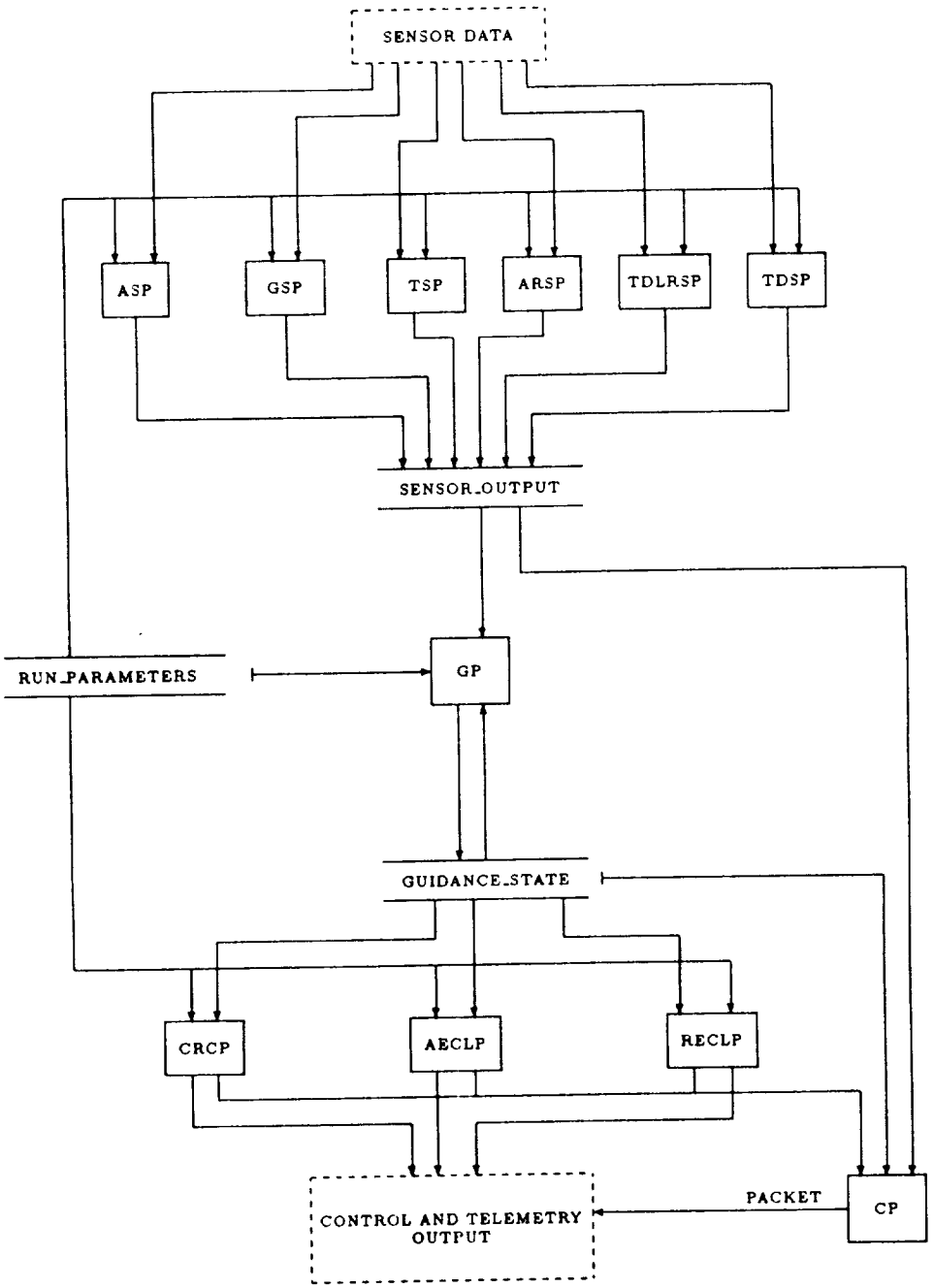


Figure 4.2: CONTROL 2. RUN_GCS

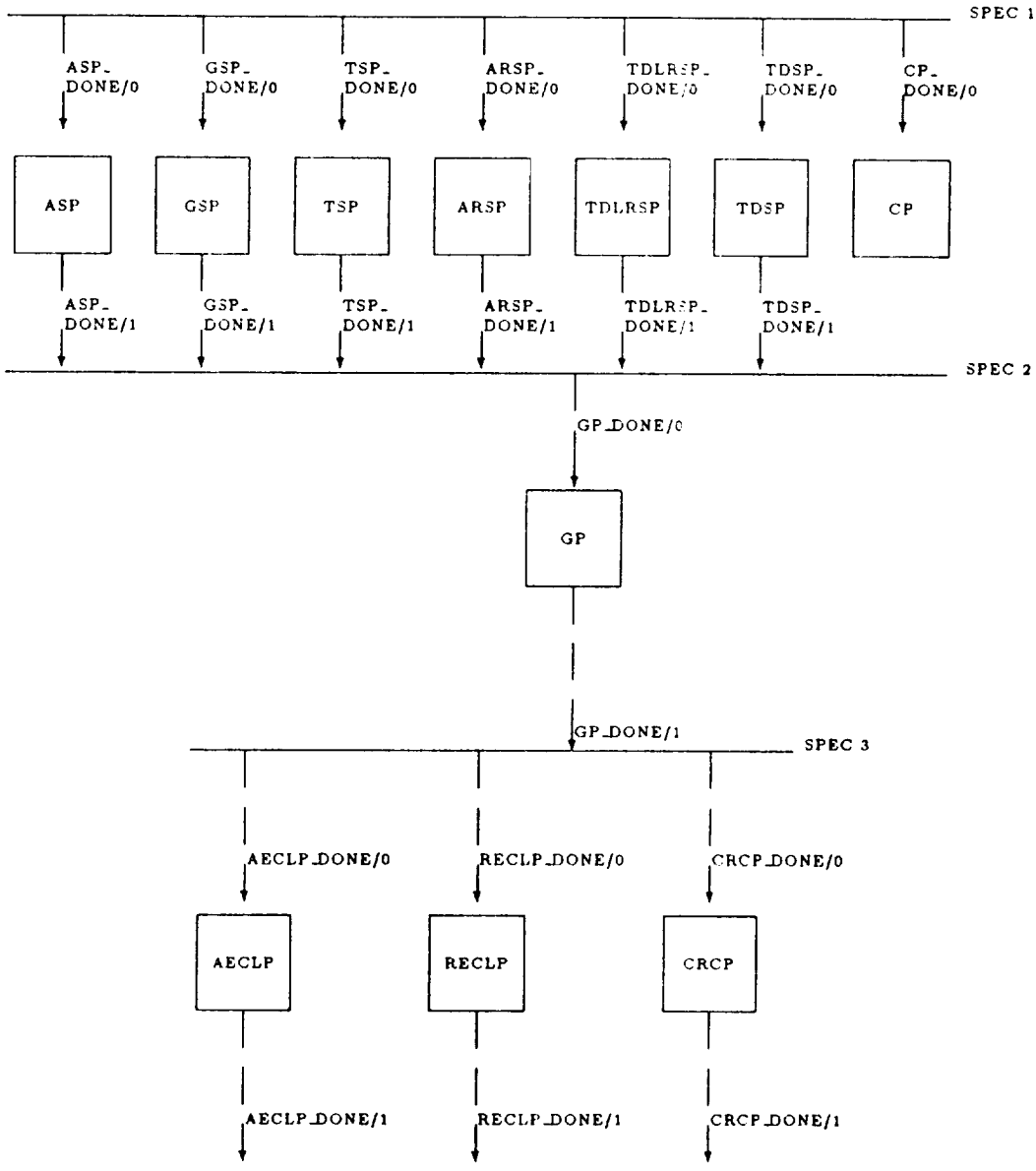


Table 4.1: CONTROL 2. RUN_GCS - SPECIFICATION 1

SCHEDULING	
Sensor Processing Sub-Frame	"I"
ARSP	2
ASP	1
CP	1
GSP	1
TDLRSP	2
TDSP	5
TSP	2
Guidance Processing Sub-Frame	"I"
CP	1
GP	1
Control Law Processing Sub-Frame	"I"
AECLP	1
CP	1
CRCP	5
RECLP	1

Above is a table listing each process in the GCS according to the subframe where they should be executed. A number "I" is located along with the process name. This number indicates that the process should be executed every "Ith" frame. Note that all processes 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. There are minor sequencing constraints to be imposed upon the modules in each subframe. During the sensor processing subframe, TSP should be executed before any of the other modules, and CP should be executed last. In the guidance and control subframes, CP should be executed after the other modules. Lastly, during the control subframe, AECLP needs to be executed before CRCP. All modules not specified here may be executed in any order within their subframes. On the first, and subsequent, calls to rendezvous, FRAME_COUNTER will be returned to the application containing the correct value for operation. The value in FRAME_COUNTER should be compared to the numbers listed below to determine if a process should be executed. As an example, ARSP 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. Chapter 6 provides additional information on timing requirements.

Table 4.2: CONTROL 2. RUN_GCS - SPECIFICATION 2

INPUT			OUTPUT		Activate
"I"	Control Variable	Value	Control Variable	Value	Process
1	ARSP_DONE	1	GP_DONE	0	GP
1	ASP_DONE	1			
1	GSP_DONE	1			
1	TSP_DONE	1			
1	TDLRSP_DONE	1			
1	TDSP_DONE	1			

Table 4.3: CONTROL 2. RUN_GCS - SPECIFICATION 3

INPUT			OUTPUT		Activate
"I"	Control Variable	Value	Control Variable	Value	Process
1	GP_DONE	1	AECLP_DONE	0	AECLP
1	GP_DONE	1	RECLP_DONE	0	RECLP
5	GP_DONE	1	CHUTE_RELEASED	0 or 1	CRCP

5. LEVEL 3 SPECIFICATION

2.1 AECLP - Axial Engine Control Law Processing

PURPOSE The AECLP module 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

A_ACCELERATION	AE_STATUS
AE_SWITCH	AE_TEMP
CHUTE_RELEASED	DELTA_T
FRAME_COUNTER	FRAME_ENGINES_IGNITED
FULL_UP_TIME	CONTOUR_CROSSED
ENGINES_ON_ALTITUDE	GA
GAX	GP_ALTITUDE
GP_ROTATION	GP_VELOCITY
GP1	GP2
GPY	GQ
GR	GV
GVE	GVEI
GVI	GW
GWI	OMEGA
PE_INTEGRAL	PE_MAX
PE_MIN	TE_INTEGRAL
TE_INIT	TE_LIMIT
TE_MAX	TE_MIN
TE_DROP	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 Computation of the axial engine valve settings requires the following steps:

- DETERMINE IF AXIAL ENGINES ARE SWITCHED ON –
  If AE_SWITCH is set to OFF, then set AE_CMD = 0, set axial engine status to healthy and proceed directly to the step "COMMAND ENGINES".

- DETERMINE ENGINE TEMPERATURE –
  Engine temperature is determined according to the events in Table 5.1.

Table 5.1: DETERMINATION OF AXIAL ENGINE TEMPERATURE

Current Axial Engine Temperature	Event	Next Axial Engine Temperature
Cold	GP_ALTITUDE > Altitude to start engines	Cold
Cold	(GP_ALTITUDE ≤ Altitude to start engines) and (FRAME_COUNTER - FRAME_ENGINES_IGNITED) · DELTA_T < FULL_UP_TIME	Warming_up
Warming_up	(GP_ALTITUDE ≤ Altitude to start engines) and (FRAME_COUNTER - FRAME_ENGINES_IGNITED) · DELTA_T ≥ FULL_UP_TIME	Hot

- COMPUTE LIMITING ERRORS -

  - Compute limiting vehicle pitch (P_e^L), yaw (Y_e^L), and thrust (T_e^L) errors using the following Proportional-Integral-Derivative (P-I-D) control law: $\epsilon = a_0 + a_1\theta + a_2 XX\_INTEGRAL$. In these equations, $XX\_INTEGRAL = XX\_INTEGRAL + \int_{t_0}^{t} \theta dt$ and XX is to be replaced with one of the following; PE, YE, or TE depending on the type of error being calculated. Note

that t_o is the beginning of the time step and t is the end of the time step; and the integration for PE and YE begins when the engines are turned on, while the integration for TE begins when the engines get hot. The terms of this control law, used for calculating P_e^L and Y_e^L, are given in Table 5.2, where p_v, q_v, and r_v are input as elements of GP_ROTATION; $\dot{x}_v$, $\dot{y}_v$, and $\dot{z}_v$ are input in GP_VELOCITY; $\ddot{x}_v$ is input in A_ACCELERATION; and the gains are input as specified. If either PE_MIN $> P_e^L$ or PE_MAX $< P_e^L$, then P_e^L should be set to either PE_MIN or PE_MAX respectively. Similarly, boundary values hold for Y_e^L and T_e^L.

The variable TE_LIMIT is provided for use in calculating T_e^L since the equation for T_e^L is differential in nature, thus requires an input value for each time step and it is also bounded by a maximum and minimum value. TE_LIMIT should be calculated as given in Table 5.2. Then T_e^L is set to TE_MAX if TE_LIMIT is greater than or equal to TE_MAX; or T_e^L is set to TE_MIN if TE_LIMIT is less than or equal to TE_MIN; or finally, if TE_LIMIT is within the region bounded by TE_MAX and TE_MIN, T_e^L is set equal to TE_LIMIT. Thus TE_LIMIT is not bounded by TE_MAX and TE_MIN, but contains a valid value for use as an input to the calculations during the next frame.

Table 5.2: P_e^L, Y_e^L, and T_e^L CONTROL LAW COEFFICIENTS

ϵ	a_0	a_1	a_2	θ
P_e^L	GQ $\cdot q_v$	GW	GWI	$\dot{z}_v/\dot{x}_v$
Y_e^L	-GR $\cdot r_v$	GV	GVI	$\dot{y}_v/\dot{x}_v$
$\frac{\frac{dTE\_LIMIT}{dt}+OMEGA \cdot TE\_LIMIT}{GA}$	$-GAX \cdot \ddot{x}_v$	GVE	GVEI	VELOCITY_ERROR

- COMPUTE PITCH, YAW, AND THRUST ERRORS
  - Pitch, yaw, and thrust errors should then be calculated according to Table 5.3.
- COMPUTE AXIAL ENGINE VALVE SETTINGS -
  Given a pitch, yaw, and thrust error, (P_e , Y_e , T_e), the valve settings (AE_CMD) for each of the three main engines are calculated as:

Table 5.3: DETERMINATION OF ERROR TERMS

AE_ SWITCH	CHUTE_ RELEASED	CONTOUR_ CROSSED	P_e	Y_e	T_e
1	1	1	P_e^L	Y_e^L	T_e^L
1	1	0	P_e^L	Y_e^L	TE_DROP
1	0	0,1	$GQ \cdot q_v$	$-GR \cdot r_v$	TE_INIT
0	0,1	0,1	0	0	0

$$INTERNAL\_CMD := \begin{pmatrix} GP1 & 0 & 1 \\ GP2 & -GPY & 1 \\ GP2 & GPY & 1 \end{pmatrix} \times \begin{pmatrix} P_e \\ Y_e \\ T_e \end{pmatrix}$$

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 should be as follows:

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

with INTERNAL_CMD between 0 and 1.0 being converted **linearly** (with truncation) to a value of AE_CMD between 0 and 127.

- COMMAND ENGINES - 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)

- SET AXIAL ENGINE STATUS TO HEALTHY

2.2 ARSP - Altimeter Radar Sensor Processing

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

INPUT	AR_ALTITUDE AR_FREQUENCY K_ALT	AR_COUNTER AR_STATUS

OUTPUT	AR_ALTITUDE K_ALT	AR_STATUS

PROCESS Note that AR_ALTITUDE, AR_STATUS, and K_ALT are five element arrays containing the present value as well as the previous four values of altitude, status, and state respectively. Also note that as the new value is calculated, it is placed into the "zeroth" position; the others are rotated to the (i+1st) position in the array, where i is the index of the current position for that value. The value whose index is out of bounds is dropped. 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 10-bit digital counter (AR_COUNTER) is started as the radar pulse is transmitted. The counter increments AR_FREQUENCY times per second. The 10-bit value is placed into the lower ten bits of the 16-bit counter.

- READ SENSOR -
  Upon return from the call to GCS_SIM_RENDEZVOUS prior to this subframe, an updated value will have been put into AR_COUNTER. This value should be used for the present iteration of ARSP.

- DETERMINE ALTITUDE - When the altitude is calculated, rotate the AR_ALTITUDE array down by one place, and put the calculated value in the "zeroth" position of AR_ALTITUDE.

  - ECHO RECEIVED -
    Convert the AR_COUNTER value to a distance to be returned

in the variable AR_ALTITUDE by the following equation:

$$AR\_ALTITUDE = \frac{AR\_COUNTER \cdot 3E8\frac{m}{sec}}{AR\_FREQUENCY \cdot 2}$$

- ECHO NOT RECEIVED -
  If an echo is not received, AR_COUNTER will return all ones. To smooth the estimate of altitude, fit a third-order polynomial to the previous four values of AR_ALTITUDE. This polynomial fit should then be used to extrapolate an altitude value for the current time step. This extrapolation should be done even if one or more previous values of AR_STATUS is unhealthy. In the case of one or more unhealthy values, the extapolated value will not be used, but should be calculated.

- SET ALTIMETER RADAR STATUS -
  The values in AR_STATUS and K_ALT should be rotated and when they are calculated, the new values should be placed in the "zero" position as were the altitude values. Set the altimeter status according to Table 5.4 and determine the value of K_ALT for use in the GUIDANCE PROCESSOR.

Table 5.4: USE OF STATUS IN CALCULATION OF ALTITUDE

CONDITION	AR_STATUS	ACTION
Echo returned	Healthy	K_ALT=1
No echo returned but used healthy values in polynomial	Failed	K_ALT=1
No echo returned and one or more failed values in the previous four time steps	Failed	K_ALT=0

This table is used to determine the method to calculate the altitude. Each of the possible states of the radar is listed along with the appropriate actions for that situation.

2.3 ASP - Accelerometer Sensor Processing

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 in A_COUNTER is smaller than that of A_BIAS, the acceleration is negative. The Acceleration Sensor Processing (ASP) module 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 three steps:

- READ ACCELEROMETER -
  Upon return from the call to GCS_SIM_RENDEZVOUS prior to this subframe, an updated value will have been put into A_COUNTER. This value should be used for the present iteration of ASP.

- 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.

  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.

That is,

$$A\_GAIN(i) := A\_GAIN\_0(i) + (G1 \cdot ATMOSPHERIC\_TEMP) + (G2 \cdot ATMOSPHERIC\_TEMP^2)$$

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

Where A_GAIN_0 is the standard gain. A_GAIN_0, A_BIAS, G1, and G2 are set during initialization mode. The equation for measured acceleration then becomes:

$$A\_ACCELERATION\_M(i) = A\_BIAS(i) + A\_GAIN(i) * A\_COUNTER(i)$$

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 following multiplier matrix, which is based on small angle approximations, corrects for this misalignment. The matrix is used for transforming the measured acceleration data into the true vehicle accelerations.

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

and

$$A\_ACCELERATION = ALPHA\_MATRIX \times A\_ACCELERATION\_M$$

The input variable, ALPHA_MATRIX, defines the values of the α's in this multiplier matrix. For example, ALPHA_MATRIX(1,3), 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. These misalignment angles are set during GCS initialization mode.

- DETERMINE ACCELERATIONS AND ACCELEROMETER STATUS - 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. They are to be rotated similar to those in 2.2 ARSP.

- If one or more of the previous three values of status is unhealthy, use the present value of A_ACCELERATION and set the current value of A_STATUS to healthy.
- If the previous values of status are healthy, check for extreme values and set A_STATUS and A_ACCELERATION according to the equations below. The accelerometer processing includes filtering of the calculated accelerations along each axis (i.e filtering of $(\ddot{x}_v, \ddot{y}_v, \ddot{z}_v)_t$), ignoring or eliminating calculated accelerations which are out of range. To effect this filtering, the means and standard deviations of each component of these accelerations 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 let

$$\hat{\mu} = \sum_{i=t-3}^{t-1} \frac{\ddot{x}_{v(i)}}{3}$$

be the current sample mean and

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

be the current sample standard deviation. If

$$|\hat{\mu} - \ddot{x}_v(t)| > A\_SCALE \cdot \hat{\sigma}$$

then set

$$\ddot{x}_v(t) := \hat{\mu}$$

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.

* If the calculation for the current time step for any component differs from the mean by more than A_SCALE times the standard deviation, then that component should be replaced by its current mean and A_STATUS should be set to unhealthy.
* If the calculated value of acceleration is within the specified range of the mean, use the calculated value and place it into A_ACCELERATION. Then set the status to healthy.

2.4 CP - Communications Processing

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

INPUT	AE_CMD	C_STATUS
	COMM_SYNC_PATTERN	FRAME_COUNTER
	GUIDANCE_STATE	RE_CMD
	SENSOR_OUTPUT	

OUTPUT	C_STATUS	PACKET

PROCESS The data packet, PACKET, prepared for transmission is organized to sequentially contain a synchronization pattern, a sequence number, checksum information, new sample mask, and the data itself.

The construction of the packet requires five 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 repeat after the 255th packet and can be calculated based on the FRAME_COUNTER and the subframe where the present call to CP was made.
  - 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 sub-frame should be marked in the mask and transmitted. 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. A position should represent each variable contained in either GUIDANCE_STATE or SENSOR_OUTPUT in addition to AE_CMD and RE_CMD. These variables should be arranged as shown in Table 5.5.

- 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. 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 G_ROTATION) and since these terms can never have "new" values, they should not be transmitted.
- 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.

- SEND PACKET - The data packet created, PACKET, will automatically be transmitted during the next call to RENDEZVOUS.
- SET COMMUNICATOR STATUS TO HEALTHY

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	

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.

2.5 CRCP - Chute Release Control Processing

PURPOSE The CRCP module implements the release of the parachute which is attached at 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 at the same value and this signal will be automatically transmitted to the chute release mechanism during the next call to the rendezvous routine provided to the user (See Appendix B. Implementation Notes). If the chute has not been released, the engine temperature will determine whether or not to release the chute. If the engines are hot (i.e. AE_TEMP is HOT), then release the chute by setting CHUTE_RELEASED to 1.

2.6 GSP - Gyroscope Sensor Processing

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) module 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	G_STATUS

OUTPUT

G_ROTATION	G_STATUS

PROCESS The output from each of the gyroscope (G_COUNTER) is a 16-bit quantity 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 the values of G_ROTATION so the present values are in the "zeroth" position of the time dimension and the previous values are rotated to the (i+1st) position in the array, where i is the index of the current position for that value. The value whose index is out of bounds is dropped.
- 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.

where G_GAIN_0, G3, and G4 are set during GCS initialization mode.

- 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.

2.7 GP - Guidance Processing

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	CONTOUR_ALTITUDE
CONTOUR_CROSSED	CONTOUR_VELOCITY
DELTA_T	DROP_HEIGHT
ENGINES_ON_ALTITUDE	FRAME_COUNTER
GP_ALTITUDE	GP_ATTITUDE
GP_PHASE	GP_VELOCITY
GRAVITY	G_ROTATION
K_ALT	K_MATRIX
RE_SWITCH	TD_SENSED
TDLR_VELOCITY	TDS_STATUS

OUTPUT

AE_SWITCH	CONTOUR_CROSSED
FRAME_ENGINES_IGNITED	GP_ALTITUDE
GP_ATTITUDE	GP_PHASE
GP_ROTATION	GP_VELOCITY
RE_SWITCH	VELOCITY_ERROR

ARRAYS The variables GP_ATTITUDE, GP_ALTITUDE, and GP_VELOCITY are five element arrays in each of their spatial dimensions and contain enough previous values to provide the required history for integration in updating the vehicle and guidance states. The most recent values are in the array locations indexed by the lower numbers. Thus the "zero" position represents the present values. This implies that before calculating the values for the present time step, all values in such arrays should be rotated by placing the "three" value into the "four" position, then the "two" value into the "three" position, etc. This will leave the "zero" position ready for the soon-to-be-calculated value and will discard the "four" position value.

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

- SET UP THE GP_ROTATION MATRIX - G_ROTATION contains three values: p, q, and r. These values must be placed into a 3 x 3 matrix in the correct positions for later calculations. This matrix is GP_ROTATION and is organized as follows:

$$GP\_ROTATION = \begin{pmatrix} 0 & r & -q \\ -r & 0 & p \\ q & -p & 0 \end{pmatrix}$$

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.

- CALCULATE NEW VALUES OF VELOCITY, ALTITUDE, AND ATTITUDE -

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

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

For instance:

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

where $\dot{X}$ represents the rates of change of velocity, altitude, or attitude. These are calculated according to the following formula: $\frac{d}{dt}$(variable) = $\alpha\times$ Variable + β + correction term. Table 5.8 shows the values of the variables, α, β, and the correction terms.

Note:

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.

3. All matrices are referenced with the row being the first index, the column being the second index, and time being the last if there is a time dimension.

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 OFF -

  Axial engines should:

  1. remain unchanged if GP_ALTITUDE > ENGINES_ON_ALTITUDE
  2. be set to "on" if GP_ALTITUDE $\leq$ ENGINES_ON_ALTITUDE
  3. be set to "off" if GP_ALTITUDE is $\leq$ DROP_HEIGHT
  4. be set to "off" if TD_SENSED is 1.

  Higher numbered conditions override lower numbered conditions; thus if the engines have been turned off by 3 or 4, condition 2 can never turn them on again.

  If the axial engines are turned on during this frame, FRAME_ENGINES_IGNITED should be set with the current value of FRAME_COUNTER for the later use of AECLP in determining engine temperature. FRAME_ENGINES_IGNITED will be initialized to zero, and should only be changed during the frame when the axial engines are turned on.

  Roll engines should be on unless the axial engines have been turned off due to conditions 3 or 4 above. Note that roll engines may only be turned off; they can never be turned on again even if neither condition 3 nor 4 remains valid.

  Engines are turned on or off by setting the SWITCH variables to the appropriate values.

Table 5.8: DIFFERENTIAL EQUATIONS

Variable	α	β	CorrectionTerms
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× (TDLR_VELOCITY− GP_VELOCITY)
GP_ALTITUDE	0	−GP_ATTITUDE× GP_VELOCITY	K_ALT · (AR_ALTITUDE− GP_ALTITUDE)

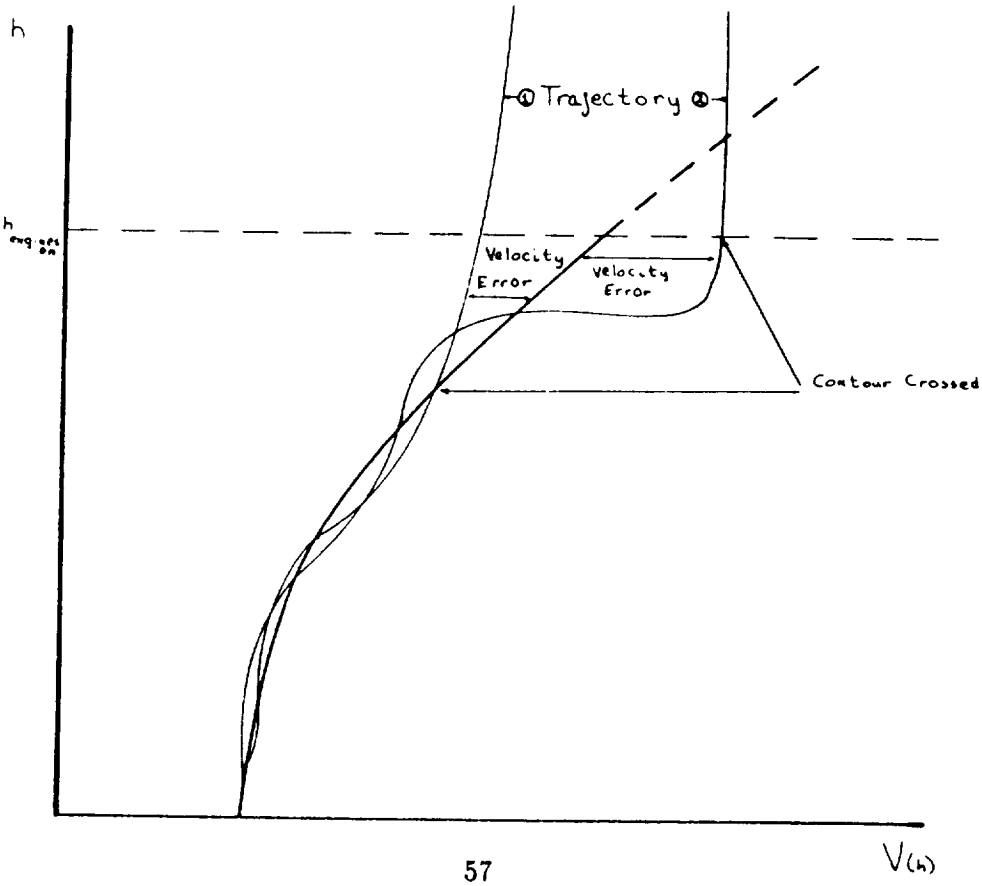
- DETERMINE VELOCITY ERROR - Calculate the difference between the velocity of the craft and the optimal velocity of the craft at the vehicle altitude (Shown in Figure 5.1.) This distance is actually a difference between two velocities and is called VELOCITY_ERROR. This error term should be calculated by finding the present altitude in CONTOUR_ALTITUDE and using interpolation, if necessary, then locating the corresponding velocity in CONTOUR_VELOCITY also using interpolation, if necessary. VELOCITY_ERROR is used in AECLP, and it is also used to set the CONTOUR_CROSSED switch. The equation for VELOCITY_ERROR is given below:

$$VELOCITY\_ERROR = |GP\_VELOCITY| - CONTOUR\_VELOCITY$$

- DETERMINE IF CONTOUR HAS BEEN CROSSED -
If CONTOUR_CROSSED has not been set and the contour has been encountered, set CONTOUR_CROSSED to 1; otherwise leave it alone.

Figure 5.1: VELOCITY ALTITUDE CONTOUR

Shown are 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. Note that 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. 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.



- DETERMINE GUIDANCE PHASE – The guidance phase (GP_PHASE) is determined according to the events in Table 5.9. These phases are based upon information that may be provided by processes other than the guidance processor.

Table 5.9: GUIDANCE PHASES

PHASE	STATE	EVENT	NEXT PHASE	NEXT STATE
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	Touch down is sensed	End GCS	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 touch down not sensed	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	End GCS	Chute Released Engines off Touch down not sensed
3	Chute released Axial Engines Hot Touch down not sensed	Touch down is sensed	End GCS	Chute Released Engines off Touch down sensed
4	Chute released Engines off Touch down not sensed	Touch down is sensed	End GCS	Chute Released Engines off Touch down sensed
4	Chute released Engines off Touch down not sensed	TDS_STATUS = failed	End GCS	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, begin Phase 2.
  - PHASE 2 : If the axial engines have become hot and the parachute has been released, begin Phase 3. If touch down is sensed, end GCS.
  - 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 TDS_STATUS is healthy, begin Phase 4. If DROP_HEIGHT is reached and TDS_STATUS is failed, send final packet, and end GCS. If touch down is sensed, send final packet, and end GCS.

- PHASE 4 : If touch down has not been sensed and TDS_STATUS is healthy, free-fall to surface. If touch down has not been sensed and TDS_STATUS is failed, send final packet and end GCS. If touch down has been sensed, send final packet and end GCS.

It should be noted that under certain conditions, the next phase is "End GCS". This means that the implementation should stop itself at the end of the present sub-frame. Thus, in all cases, a clean shutdown of GCS implementations should end just after Communications Processing during the Guidance sub-frame, but before calling rendezvous.

2.8 RECLP - Roll Engine Control Law Processing

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_STATUS	RE_SWITCH
THETA	THETA1
THETA2	

OUTPUT

RE_CMD	RE_STATUS
THETA	

PROCESS Control of the lander is achieved by generating commands as functions of the error between a given state variable and its ideal value. These errors are limited and amplified to yield control values. The transformations to accomplish this are as follows:

- DETERMINE IF ENGINES ARE ON - If RE_SWITCH is off, then RE_CMD = 0; and proceed directly to commanding engines.
- DETERMINE PULSE INTENSITY AND DIRECTION - The pulse intensity and direction is derived from the graph shown in Figure 5.2 using $(p_v)_t$. 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. As an example, THETA = THETA + (integral of roll for this step). Also 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.
- DETERMINE ROLL ENGINE COMMAND - The pulse intensity and direction is packed in the lowest three lower-order bits of the actual roll engine command, RE_CMD as shown.

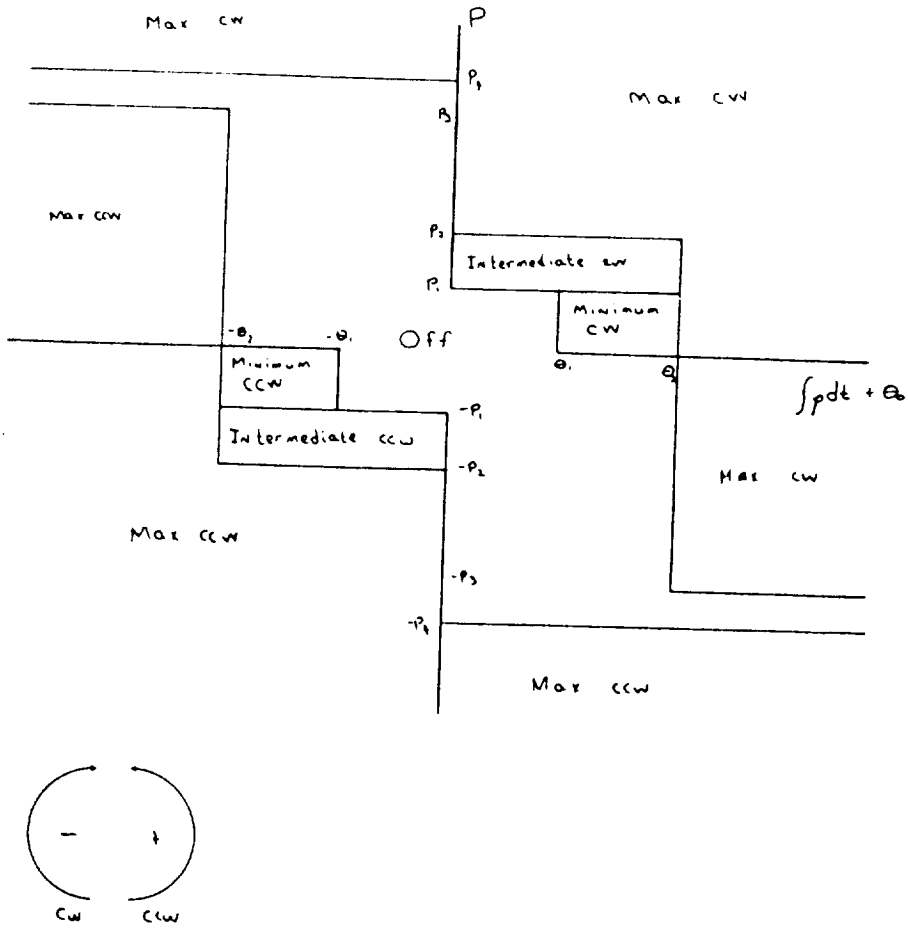
X	X	X	...	X	I	I	D
16	15	14	...	4	3	2	1

where X = unused, I = intensity, and D = direction. I and D range in values as shown in the data dictionary.

- COMMAND ENGINES -
  Once 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.

- SET ROLL ENGINE STATUS TO HEALTHY.

Figure 5.2: GRAPH FOR DERIVING ROLL ENGINE COMMANDS



2.9 TDLRSP - Touch Down Landing Radar Sensor Processing

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 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) module converts measurements of the frequency shift of each beams reflection into vehicle velocities. The receivers associated with each beam may not find a usable reflection, though. If no usable reflection is found, the receiver returns a status of beam in search mode.

INPUT

DELTA_T	FRAME_BEAM_UNLOCKED
FRAME_COUNTER	K_MATRIX
TDLR_ANGLES	TDLR_COUNTER
TDLR_GAIN	TDLR_LOCK_TIME
TDLR_OFFSET	TDLR_STATE
TDLR_STATUS	TDLR_VELOCITY

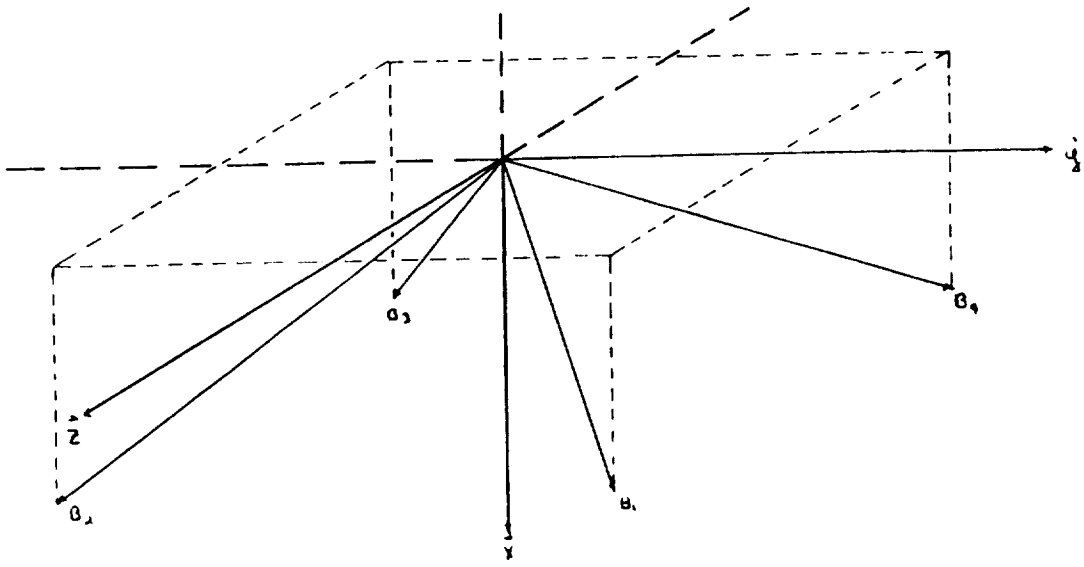
OUTPUT

FRAME_BEAM_UNLOCKED	K_MATRIX
TDLR_STATE	TDLR_STATUS
TDLR_VELOCITY	

PROCESS 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 five steps.

- ROTATE VALUES – Rearrange the values located in TDLR_VELOCITY and K_MATRIX so that each value is moved to the variable with the next larger index. Thus the values are rotated to the (i+1st) position in the array, where i is the index of the current position for that value. The value whose index is out of bounds is dropped. For example, the "zeroth" position is left empty for new values and the value that was in the "zeroth" position is now in the first position, etc. and the value that was in the fourth position is lost.

Figure 5.3: DOPPLER RADAR BEAM LOCATIONS



- DETERMINE RADAR BEAM STATES - The processing of the four radar beams depends on the state of the radar, i.e. whether or not each of the four beams is searching or in lock. If TDLR_STATE is LOCKED, and 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; TDLR_STATE should be set to UNLOCKED and FRAME_BEAM_UNLOCKED should be set to the current frame count. If the previous state of TDLR_STATE is UNLOCKED, FRAME_BEAM_UNLOCKED should be used to ignore the beam for TDLR_LOCK_TIME seconds of real time, thus determining the current value of TDLR_STATE. At the beginning of a trajectory, FRAME_BEAM_UNLOCKED will be set to zero, thus meaning that the beam has never been unlocked. If TDLR_STATE is not UNLOCKED due to the above conditions, it should be set to LOCKED.

- 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. TDLR_GAIN and TDLR_OFFSET are set during GCS initialization mode. The equation for velocity is given below.

$$BEAM\_VELOCITY(i) = TDLR\_OFFSET + TDLR\_GAIN * (TDLR\_COUNTER(i))$$

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

- AVERAGE BEAM VELOCITIES AND CONVERT TO BODY VELOCITIES - The beam velocities are resolved as specified in Table 5.10. The resolved beam velocities are then converted to vehicle body velocities using the offset angles α , β , and γ as shown in Figure 5.4. Note that the conversion from resolved beam velocities to body velocities is done with the following equations:

$$B_x = \frac{\bar{B}_x}{\cos\alpha}$$

$$B_y = \frac{\bar{B}_y}{\cos\beta}$$

$$B_z = \frac{\bar{B}_z}{\cos\gamma}$$

B_x, B_y, B_z are actually the values of the elements of TDLR_VELOCITY. Since the Guidance Processor needs to know which velocities it can use,

Table 5.10: AVERAGING DOPPLER RADAR BEAMS IN LOCK

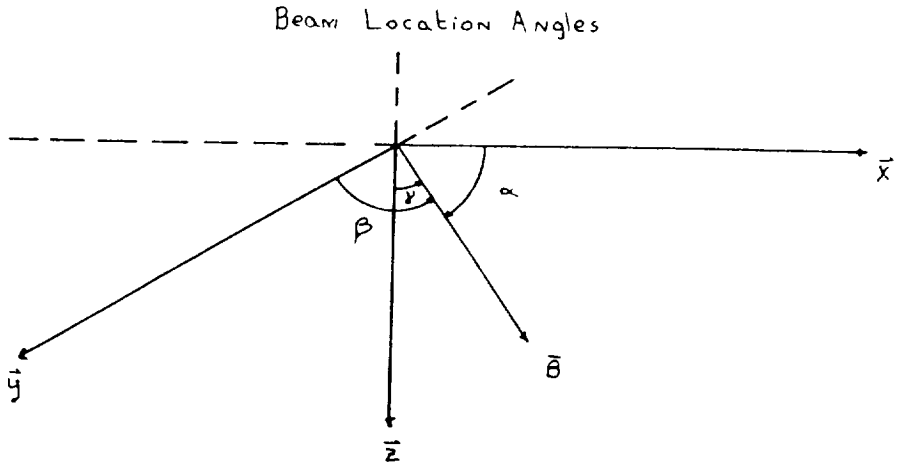
BEAMS IN LOCK	B_x	K_x	B_y	K_y	B_z	K_z
none	0	0	0	0	0	0
B_1	0	0	0	0	0	0
B_2	0	0	0	0	0	0
B_3	0	0	0	0	0	0
B_4	0	0	0	0	0	0
B_1, B_2	0	0	$(B_1 - B_2)/2$	1	0	0
B_1, B_3	$(B_1 + B_3)/2$	1	0	0	0	0
B_1, B_4	0	0	0	0	$(B_1 - B_4)/2$	1
B_2, B_3	0	0	0	0	$(B_2 - B_3)/2$	1
B_2, B_4	$(B_2 + B_4)/2$	1	0	0	0	0
B_3, B_4	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

the K_MATRIX must be defined according to the usable velocities. The following equation shows the K_MATRIX in which the variables should be replaced with a 1 if there is a usable velocity available, or a 0 if not as shown in Table 5.10.

$$K\_MATRIX = \begin{bmatrix} K_x & 0 & 0 \\ 0 & K_y & 0 \\ 0 & 0 & K_z \end{bmatrix}$$

- SET TDLR_STATUS - Set TDLR_STATUS to healthy.

Figure 5.4: DOPPLER RADAR BEAM ANGLES



2.10 TDSP - Touch Down Sensor Processing

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:

1. avoid the stirring up of dust and debris and
2. 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. It should normally return one of only two 16-bit values, all "ones" or all "zeroes". Note that this value includes setting the sign bit as well as the 15 magnitude bits.

- DETERMINE IF TOUCH DOWN HAS BEEN SENSED:
  - If all ones are returned, set TD_SENSED to 1.
  - If all zeroes are returned, set TD_SENSED to 0.
  - If any combination of "ones" and "zeroes" is returned other than all on or all off, assume that the sensor has failed due to electrical noise and set TDS_STATUS to failed. Once TDS_STATUS is set to failed, it should remain set to failed for all following frames. The normal state of the switch is all zeroes ('off'). If all of the readings for the last time step are all ones ('on') then the current processed value for the sensor is 'on', signifying touch down has been sensed. At all other times, the processed value is 'off'; thus, if the status is set to failed, the value should be set to 'not sensed', and the guidance processor should decide when the vehicle has touched down.

2.11 TSP - Temperature Sensor Processing

PURPOSE A temperature gauge on the vehicle is used to adjust the response of the accelerometers and gyroscope. The gauge contains two temperature sensing devices: a solid-state sensor and a matched pair of thermocouples. The Temperature Sensor Processing (TSP) module 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
TS_STATUS	

OUTPUT

ATMOSPHERIC_TEMP	TS_STATUS

PROCESS 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 temperature values from the solid-state sensor are highly quantized and are used to adjust the values of the other sensors when they indicate temperatures outside the range of the thermocouple pair.

The processing of SS_TEMP and THERMO_TEMP into an accurate measure of temperature (ATMOSPHERIC_TEMP) requires several steps. The steps are described below, but are not given in any particular order because the steps to be taken may vary depending upon the values of SS_TEMP and THERMO_TEMP.

- CONVERSION OF SOLID STATE TEMPERATURE (SS_TEMP) - 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 and are set during GCS initialization.
- CONVERSION OF THERMOCOUPLE PAIR TEMPERATURE (THERMO_TEMP) - The response of the thermocouple pair is cali-

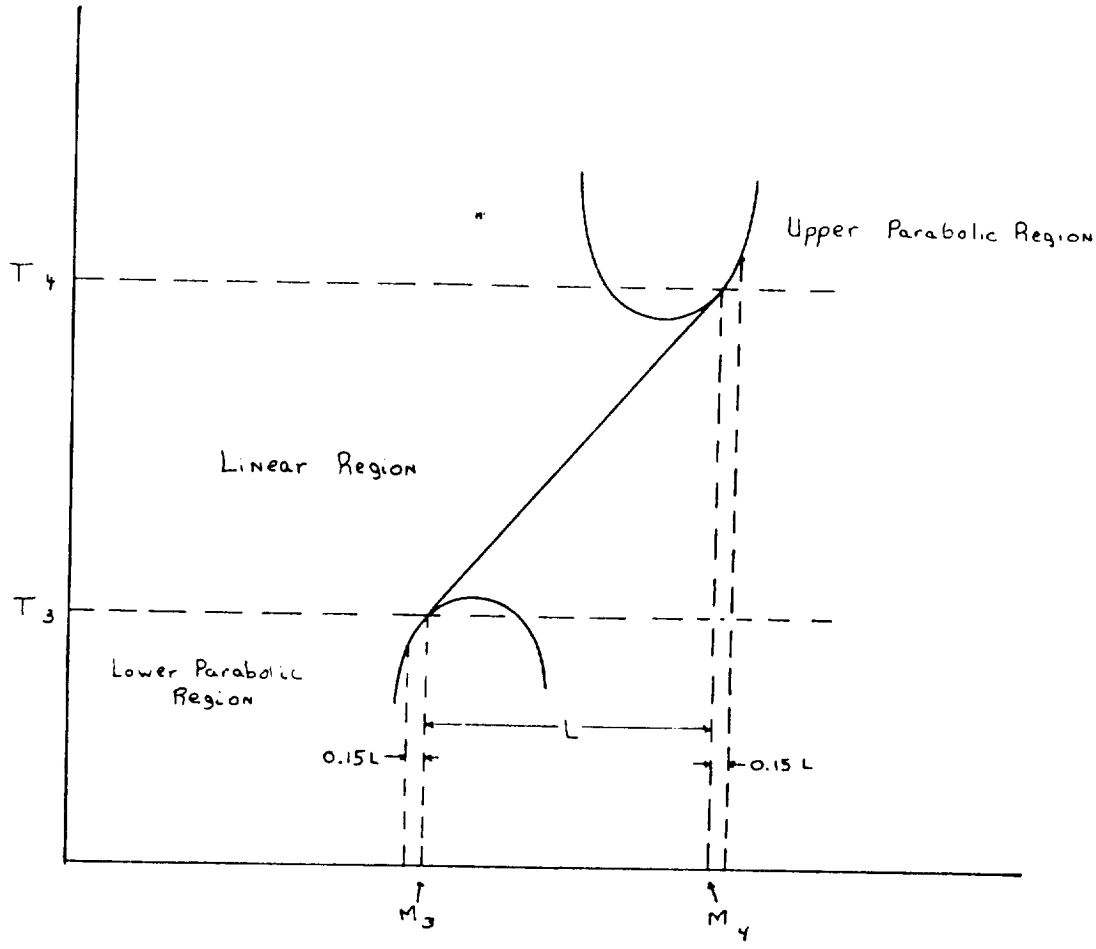
brated differently depending on the region (linear or parabolic) where the measurement lies. See Figure 5.5.

  - THERMO_TEMP lies within the 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.
  - THERMO_TEMP lies within one of the 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. 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.

- SELECT MOST ACCURATE ESTIMATE - If the temperature derived from SS_TEMP falls within the accurate temperature response zone of the thermocouple pair, (the linear as well as parabolic regions), then the value returned by the thermocouple pair should be used; otherwise, the value returned by the solid-state sensor should be used.

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

Figure 5.5: CALIBRATION OF THERMOCOUPLE PAIR



6. SYSTEM TIMING AND MEMORY SPACE REQUIREMENTS

TIMING REQUIREMENTS

The GCS must operate within certain timing constraints to be able to provide signals to the vehicle rapidly enough to properly control the system. To allow the GCS to control the vehicle at the proper rate, each module must execute within a specified time, so that all modules to be executed can complete before the end of the subframe. These execution times must be determined by the minimum time available, which is also the time that the most processes are to execute. Some processes execute at a lower frequency than others; thus for some frames, there may be processes that are not executed, leaving extra time remaining in the frame after the last process finishes and before the next subframe begins. However, there will also be frames during which all processes execute, and thus the time allocated for each module is strictly limited.

Model Time

The GCS is part of a larger simulation that consists of GCS_SIM and one or more versions of the GCS. When these two parts (GCS and GCS_SIM) are combined, they approximate the behavior of the environment around a planetary lander (wind, gravity, etc.); the physical behavior of the lander (acceleration, engine thrust, etc.); and the on-board control algorithms (GP, AECLP, etc.). Since the experiment being conducted is interested in detection of software errors, the part of the simulation under study is only the GCS. Thus GCS becomes the "model" upon which tests will be conducted. For realism, constraints in timing and memory are being placed on GCS to simulate the restricted environment of typical embedded systems aboard air/spacecraft. Thus, the constraints and requirements listed that refer to the "model" are only those limitations being placed on a single version of the GCS, and the programmers should treat them as restrictions on their code without concern for the simulator within which their code will run.

The model operates with three subframes making up each frame, and each frame executes within a period of DELTA_T. Therefore, each subframe has a duration of $\leq \frac{DELTA\_T}{3}$. Note that returning from a call to rendezvous is a signal to increment the subframe. At the end of the control law processing subframe, FRAME_COUNTER will be updated by rendezvous, and the correct value will be returned. Figure 6.1 shows an abbreviated timeline for the system.

Response Times

Software throughput timing shall not exceed the total time allotted for each frame. Synchronization points demarcate the end of each frame.

Execution timing and memory space requirements are levied against the following three sub-frames per time step which occur sequentially:

Sub-Frame I SENSOR DATA PROCESSING

Sub-Frame II GUIDANCE PROCESSING

Sub-Frame III ENGINE CONTROL PROCESSING

Figure 6.1: TYPICAL TIME LINE

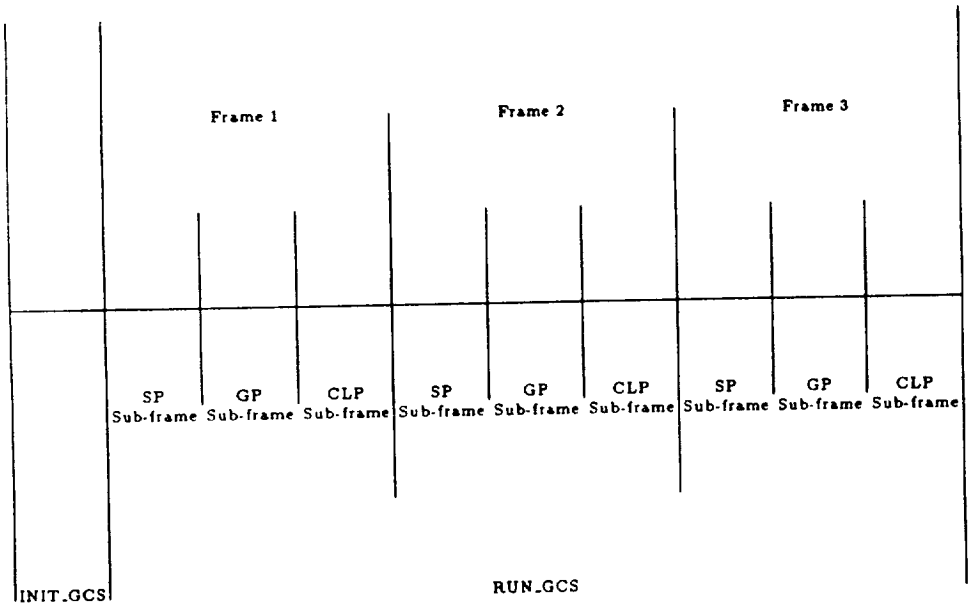


Table 6.1 depicts the timing requirements per frame.

Table 6.1: TIMING REQUIREMENTS

SUBFRAME	TIME REQUIREMENTS
I	tbd
II	tbd
III	tbd

These requirements will be determined after testing of the GCS prototype version is completed.

MEMORY SPACE REQUIREMENTS

The memory allowed for each version will include the global space needed for the required data stores, as well as some space for internal variables. It should be remembered that the applications cannot carry any global values from frame to frame except those explicitly contained within the data stores. The values of memory sizes listed in Table 6.2 include both the global space and all allowable internal space for use by the applications.

Table 6.2: MEMORY SPACE REQUIREMENTS

SUBFRAME	SPACE REQUIREMENTS
I	tbd
II	tbd
III	tbd

7. DATA REQUIREMENTS DICTIONARY

PART I. DATA ELEMENT DESCRIPTIONS

The following template has been constructed for defining the data elements referenced in this specification:

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 modules 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 permissible 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 during voting.[1]

[1] 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_ACCELERATION
DESCRIPTION: vehicle accelerations
USED IN: 2.1 AECLP, 2.3 ASP, 2.7 GP
UNITS: $\frac{meters}{sec^2}$
RANGE: [-20, 20]
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: 2.3 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 $\bar{z}$
USED IN: 2.3 ASP
UNITS: none
RANGE:[0, $2^{15} - 1$]
DATA TYPE: array (1..3) of Integer*2
ATTRIBUTE: data
DATA STORE LOCATION: EXTERNAL
ACCURACY: N/A

NAME: A_GAIN_0
DESCRIPTION: standard gain in the accelerations
USED IN: 2.3 ASP
UNITS: $\frac{m}{\frac{A^2}{count}}$
RANGE: [0, 1]
DATA TYPE: array (1..3) of real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: A_SCALE
DESCRIPTION: multiplicative constant used to determine limit on deviation accelerometer values.
USED IN: 2.3 ASP
UNITS: none
RANGE: [0, $2^{15} - 1$]
DATA TYPE: Integer*4
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: A_STATUS
DESCRIPTION: Flag indicating whether or not the accelerometers are working properly.
USED IN: 2.3 ASP, 2.4 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: [0: running of task 2.1 AECLP incomplete, 1: running of task 2.1 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: 2.1 AECLP, 2.4 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: Flag indicating whether or not axial engines are working properly.
USED IN: 2.1 AECLP, 2.4 CP
UNITS: none
RANGE: [0: Healthy, 1: Failed.]
DATA TYPE: logical*1
ATTRIBUTE: data condition
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: N/A

NAME: AE_SWITCH
DESCRIPTION: Flag indicating whether or not axial engines are turned on.
USED IN: 2.1 AECLP, 2.7 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
ACCURACY: N/A

NAME: AE_TEMP
DESCRIPTION: Temperature of axial engines when they are turned on.
USED IN: 2.1 AECLP, 2.4 CP, 2.5 CRCP, 2.7 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: 2.3 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: 2.2 ARSP, 2.4 CP, 2.7 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_COUNTER
DESCRIPTION: counter containing elapsed time since transmission of radar pulse
USED IN: 2.2 ARSP
UNITS: Cycles
RANGE: $[-1, 2^{10} - 1]$
DATA TYPE: Integer*2
ATTRIBUTE: data
DATA STORE LOCATION: EXTERNAL
ACCURACY: N/A

NAME: AR_FREQUENCY
DESCRIPTION: increment frequency of AR_COUNTER
USED IN: 2.2 ARSP
UNITS: $\frac{cycles}{second^2}$
RANGE: $[1, 10^9]$
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: 2.2 ARSP, 2.4 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: [0: running of task 2.2 ARSP incomplete, 1: running of task 2.2 ARSP complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: ASP_DONE
DESCRIPTION: Flag indicating completion of GCS.
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [0: running of task 2.3 ASP incomplete, 1: running of task 2.3 ASP complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: ATMOSPHERIC_TEMP
DESCRIPTION: atmospheric temperature
USED IN: 2.3 ASP, 2.4 CP, 2.6 GSP, 2.11 TSP
UNITS: degrees centigrade
RANGE: [-250, 250]
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: 2.4 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: 2.1 AECLP, 2.4 CP, 2.5 CRCP, 2.7 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: COMM_SYNC_PATTERN
DESCRIPTION: sixteen bit synchronization pattern
USED IN: 2.4 CP
UNITS: none
RANGE: [1101100110110010]
DATA TYPE: Integer*2
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: CONTOUR_ALTITUDE
DESCRIPTION: Altitude in Velocity-altitude contour. (the h in 'V(h)')
USED IN: 2.7 GP
UNITS: kilometers
RANGE: [0, 2]
DATA TYPE: array (1..100) of real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: CONTOUR_CROSSED
DESCRIPTION: Indicates if the velocity altitude contour has been sensed.
USED IN: 2.1 AECLP, 2.4 CP, 2.7 GP
UNITS: none
RANGE: [0:= contour not sensed, 1:= contour sensed]
DATA TYPE: logical*1
ATTRIBUTE: data condition
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: N/A

NAME: CONTOUR_VELOCITY
DESCRIPTION: Velocity in Velocity-altitude contour. (the V in 'V(h)')
USED IN: 2.7 GP
UNITS: $\frac{kilometers}{second}$
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 2.4 CP task.
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [0: running of task 2.4 CP incomplete, 1: running of task 2.4 CP complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: CRCP_DONE
DESCRIPTION: Flag indicating completion of 2.5 CRCP task.
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [0: running of task 2.5 CRCP incomplete, 1: running of task 2.5 CRCP complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: DELTA_T
DESCRIPTION: Time step duration
USED IN: 2.1 AECLP, 2.7 GP, 2.8 RECLP, 2.9 TDLRSP
UNITS: seconds
RANGE: [0, 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: 2.7 GP
UNITS: meters
RANGE: [0, 100]
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: 2.1 AECLP, 2.7 GP
UNITS: meters
RANGE: [0, 2000]
DATA TYPE: real*8
ATTRIBUTE: data condition
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: 2.9 TDLRSP
UNITS: none
RANGE: [0, $2^{31} - 1$]
DATA TYPE: array (1..4) of Integer*4
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE
ACCURACY: TBD

NAME: FRAME_COUNTER
DESCRIPTION: Counter containing the number of the present frame
USED IN: 2.1 AECLP, 2.4 CP, 2.7 GP, 2.9 TDLRSP
UNITS: none
RANGE: [1, $2^{31} - 1$]
DATA TYPE: Integer*4
ATTRIBUTE: data
DATA STORE LOCATION: EXTERNAL
ACCURACY: TBD

NAME: FRAME_ENGINES_IGNITED
DESCRIPTION: Variable containing the number of the frame during which the engines were ignited
USED IN: 2.1 AECLP, 2.7 GP
UNITS: none
RANGE: [0, $2^{31} - 1$]
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: 2.1 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: 2.3 ASP
UNITS: $\frac{meters}{second^2 / degreeC}$
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: 2.3 ASP
UNITS: $\frac{meters}{second^2 / degreeC^2}$
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: 2.6 GSP
UNITS: $\frac{radians}{second / degreeC}$
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: 2.6 GSP
UNITS: $\frac{radians}{second / degreeC^2}$
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: 2.6 GSP
UNITS: none
RANGE: $[-(2^{14} - 1), 2^{14} - 1]$
DATA TYPE: array (1..3) of Integer*2
ATTRIBUTE: data
DATA STORE LOCATION: EXTERNAL
ACCURACY: N/A

NAME: G_GAIN_0
DESCRIPTION: standard gain in vehicle rotation rates as measured by the gyroscopes
USED IN: 2.6 GSP
UNITS: $\frac{radians}{second / count}$
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: 2.6 GSP
UNITS: $\frac{radians}{sec}$
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_ROTATION
DESCRIPTION: vehicle rotation rates
USED IN: 2.4 CP, 2.6 GSP, 2.7 GP, 2.8 RECLP
UNITS: $\frac{radians}{sec}$
RANGE: [-5.0, 5.0]
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: 2.4 CP, 2.6 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: 2.1 AECLP
UNITS: $\frac{seconds}{meter}$
RANGE: [0, 50]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GAX
DESCRIPTION: gain
USED IN: 2.1 AECLP
UNITS: none
RANGE: [0, 15000]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GP1
DESCRIPTION: gain
USED IN: 2.1 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: 2.1 AECLP
UNITS: none
RANGE: [-5, 5]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GP_ALTITUDE
DESCRIPTION: altitude as seen by guidance processor
USED IN: 2.1 AECLP, 2.4 CP, 2.7 GP
UNITS: meters
RANGE: [0, 2500]
DATA TYPE: array (0..4) of real*8
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: TBD

NAME: GP_ATTITUDE
DESCRIPTION: attitude as seen by guidance processor
USED IN: 2.4 CP, 2.7 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 2.7 GP task.
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [0: running of task 2.7 GP incomplete, 1: running of task 2.7 GP complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: GP_PHASE
DESCRIPTION: phase of operation as seen by guidance processor
USED IN: 2.4 CP, 2.7 GP
UNITS: none
RANGE: [1, 4]
DATA TYPE: integer*4
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: TBD

NAME: GP_ROTATION
DESCRIPTION: rotation rates as determined by the guidance processing module
USED IN: 2.1 AECLP, 2.4 CP, 2.7 GP
UNITS: $\frac{radians}{sec}$
RANGE: [-5, 5]
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: 2.1 AECLP, 2.4 CP, 2.7 GP
UNITS: $\frac{meters}{sec}$
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: 2.1 AECLP
UNITS: none
RANGE: [-5, 5]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GQ
DESCRIPTION: gain
USED IN: 2.1 AECLP
UNITS: seconds
RANGE: [-5, 5]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GR
DESCRIPTION: gain
USED IN: 2.1 AECLP
UNITS: seconds
RANGE: [-5, 5]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GRAVITY
DESCRIPTION: gravity of planet
USED IN: 2.7 GP
UNITS: $\frac{meters}{second^2}$
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 2.6 GSP task.
USED IN: 2. RUN_GCS
UNITS: Binary
RANGE: [0: running of task 2.6 GSP incomplete, 1: running of task 2.6 GSP complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: GUIDANCE_STATE
DESCRIPTION: Data store containing all the status, state, and sensed variables in alphabetical order.
USED IN: 2.1 AECLP, 2.2 ARSP, 2.3 ASP, 2.4 CP, 2.5 CRCP, 2.6 GSP, 2.7 GP, 2.8 RECLP, 2.9 TDLRSP, 2.10 TDSP, 2.11 TSP UNITS: N/A
RANGE: N/A
DATA TYPE: common
ATTRIBUTE: data store
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: N/A

NAME: GV
DESCRIPTION: gain
USED IN: 2.1 AECLP
UNITS: $\frac{seconds}{meter}$
RANGE: [-5, 5]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GVE
DESCRIPTION: gain
USED IN: 2.1 AECLP
UNITS: /second
RANGE: $[-10^4, 10^4]$
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GVEI
DESCRIPTION: gain
USED IN: 2.1 AECLP
UNITS: $/second^2$
RANGE: [-5, 5]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GVI
DESCRIPTION: gain
USED IN: 2.1 AECLP
UNITS: /meter
RANGE: [-5, 5]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GW
DESCRIPTION: gain
USED IN: 2.1 AECLP
UNITS: $\frac{seconds}{meter}$
RANGE: [-5, 5]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: GWI
DESCRIPTION: gain
USED IN: 2.1 AECLP
UNITS: /meter
RANGE: [-5, 5]
DATA TYPE: 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: [0: initialization incomplete, 1: initialization complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: INTERNAL_CMD
DESCRIPTION: Real vector containing the command to be sent to the axial engines
USED IN: 2.1 AECLP
UNITS: none
RANGE: [-5, 5]
DATA TYPE: array (1..3) of real*8
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE
ACCURACY: TBD

NAME: K_ALT
DESCRIPTION: Determines use of altimeter radar by guidance processor
USED IN: 2.2 ARSP, 2.4 CP, 2.7 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: 2.4 CP, 2.7 GP, 2.9 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: 2.11 TSP
UNITS: none
RANGE: [0, $2^{15} - 1$]
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: 2.11 TSP
UNITS: none
RANGE: [0, $2^{15} - 1$]
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: 2.11 TSP
UNITS: none
RANGE: [0, $2^{15} - 1$]
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: 2.11 TSP
UNITS: none
RANGE: [0, $2^{15} - 1$]
DATA TYPE: Integer*2
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: OMEGA
DESCRIPTION: gain of angular velocity
USED IN: 2.1 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: 2.8 RECLP
UNITS: radians/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: 2.8 RECLP
UNITS: radians/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: 2.8 RECLP
UNITS: radians/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: 2.8 RECLP
UNITS: radians/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: 2.4 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_INTEGRAL
DESCRIPTION: Integral portion of Pitch error equation
USED IN: 2.1 AECLP, 2.4 CP
UNITS: meters
RANGE: [-1000, 1000]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: TBD

NAME: PE_MAX
DESCRIPTION: Maximum pitch error tolerable
USED IN: 2.1 AECLP
UNITS: none
RANGE: [0, 1]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: PE_MIN
DESCRIPTION: Minimum pitch error tolerable.
USED IN: 2.1 AECLP
UNITS: none
RANGE: [-1, 0]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: RE_CMD
DESCRIPTION: roll engine command
USED IN: 2.4 CP, 2.8 RECLP
UNITS: none
RANGE: 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: 2.4 CP, 2.8 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: INIT_GCS, 2.7 GP
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 2.8 RECLP task.
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [0: running of task 2.8 RECLP incomplete, 1: running of task 2.8 RECLP complete]
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: [0: running of GCS incomplete, 1: running of GCS complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: RUN_PARAMETERS
DESCRIPTION: Data store containing all the run parameters in alphabetical order.
USED IN: 2.2 ARSP, 2.3 ASP, 2.6 GSP, 2.9 TDLRSP, 2.10 TDSP, 2.11 TSP
UNITS: N/A
RANGE: N/A
DATA TYPE: common
ATTRIBUTE: data store
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: SENSOR_OUTPUT
DESCRIPTION: Data store containing all the sensor output in alphabetical order.
USED IN: 2.2 ARSP, 2.3 ASP, 2.4 CP 2.6 GSP, 2.9 TDLRSP, 2.10 TDSP, 2.11 TSP
UNITS: N/A
RANGE: N/A
DATA TYPE: common
ATTRIBUTE: data store
DATA STORE LOCATION: SENSOR_OUTPUT
ACCURACY: N/A

NAME: SS_TEMP
DESCRIPTION: Solid state temperature data
USED IN: 2.11 TSP
UNITS: none
RANGE: $[0, 2^{15} - 1]$
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: 2.11 TSP
UNITS: degrees Centigrade
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: 2.11 TSP
UNITS: degrees Centigrade
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: 2.11 TSP
UNITS: degrees Centigrade
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: 2.11 TSP
UNITS: degrees Centigrade
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: 2.10 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: 2.4 CP, 2.7 GP, 2.10 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_ANGLES
DESCRIPTION: vector of doppler radar beam offset angles (i.e., α, β, γ)
USED IN: 2.9 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: 2.9 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_GAIN
DESCRIPTION: gain in doppler radar beam
USED IN: 2.9 TDLRSP
UNITS: $\frac{meters}{second}$ / $\frac{count}{}$
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: 2.9 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: 2.9 TDLRSP
UNITS: $\frac{meters}{second}$
RANGE: [-100, 0]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: TDLR_STATE
DESCRIPTION: state of the touch down landing radar beams.
USED IN: 2.4 CP, 2.9 TDLRSP
UNITS: none
RANGE: [0: Beam out of Lock, 1: Beam in lock]
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: 2.4 CP, 2.9 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: 2.4 CP, 2.7 GP, 2.9 TDLRSP
UNITS: $\frac{meters}{second}$
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 2.9 TDLRSP task.
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [0: running of task 2.11 TDLRSP incomplete,
1: running of task 2.10 TDSP complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: TDLRSP_SWITCH
DESCRIPTION: Flag indicating
whether or not the touch down landing
radar sensor processor is turned on.
USED IN: 1. INIT_GCS
UNITS: none
RANGE: [0: processor is off,
1: processor is on.]
DATA TYPE: logical*1
ATTRIBUTE: data condition
DATA STORE LOCATION: GUIDANCE
ACCURACY: N/A

NAME: TDSP_DONE
DESCRIPTION: Flag indicating
completion of 2.10 TDSP task.
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [0: running of task 2.10 TDSP incomplete,
1: running of task 2.10 TDSP complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: TDSP_SWITCH
DESCRIPTION: Flag indicating
whether or not the touch down sensor
is turned on
USED IN: 0 GCS
UNITS: none
RANGE: [0: touch down sensor is off,
1: touch down sensor is on.]
DATA TYPE: logical*1
ATTRIBUTE: data condition
DATA STORE LOCATION: GUIDANCE
ACCURACY: N/A

NAME: TDS_STATUS
DESCRIPTION: status of the touch down sensor
USED IN: 2.4 CP, 2.7 GP, 2.10 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: 2.1 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: 2.1 AECLP
UNITS: none
RANGE: [-2, 2]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: TE_INTEGRAL
DESCRIPTION: Integral portion of Thrust
error equation
USED IN: 2.1 AECLP, 2.4 CP
UNITS: meters
RANGE: [-1000, 1000]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: TBD

NAME: TE_LIMIT
DESCRIPTION: Limiting thrust error
USED IN: 2.1 AECLP
UNITS: none
RANGE: [-10000, 10000]
DATA TYPE: real*8
ATTRIBUTE: Data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: TBD

NAME: TE_MAX
DESCRIPTION: Maximum thrust error permissible
USED IN: 2.1 AECLP
UNITS: none
RANGE: [-2, 2]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: TE_MIN
DESCRIPTION: Minimum thrust error tolerable.
USED IN: 2.1 AECLP
UNITS: none
RANGE: [-2, 2]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: THERMO_TEMP
DESCRIPTION: thermocouple pair temperature
USED IN: 2.11 TSP
UNITS: none
RANGE: [0, $2^{15} - 1$]
DATA TYPE: Integer*2
ATTRIBUTE: data
DATA STORE LOCATION: EXTERNAL
ACCURACY: N/A

NAME: THETA
DESCRIPTION: initial pulse angle
USED IN: 2.8 RECLP
UNITS: radians
RANGE: $[-\pi, \pi]$
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE
ACCURACY: TBD

NAME: THETA1
DESCRIPTION: pulse angle boundary
USED IN: 2.8 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: 2.8 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: 2.4 CP, 2.11 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 2.11 TSP task
USED IN: 2. RUN_GCS
UNITS: none
RANGE: [0: running of task 2.11 TSP incomplete, 1: running of task 2.11 TSP complete]
DATA TYPE: logical*1
ATTRIBUTE: control
DATA STORE LOCATION: none
ACCURACY: N/A

NAME: VELOCITY_ERROR
DESCRIPTION: Distance from velocity-altitude contour. (Difference in velocities from actual to desired on contour.
USED IN: 2.1 AECLP, 2.4 CP, 2.7 GP
UNITS: $\frac{meters}{second}$
RANGE: [-1500, 1500]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: TBD

NAME: YE_INTEGRAL
DESCRIPTION: Integral portion of Yaw error equation
USED IN: 2.1 AECLP, 2.4 CP
UNITS: meters
RANGE: [-1000, 1000]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: GUIDANCE_STATE
ACCURACY: TBD

NAME: YE_MAX
DESCRIPTION: Maximum yaw error permissible
USED IN: 2.1 AECLP
UNITS: none
RANGE: [-1, 1]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

NAME: YE_MIN
DESCRIPTION: Minimum yaw error tolerable.
USED IN: 2.1 AECLP
UNITS: none
RANGE: [-1, 1]
DATA TYPE: real*8
ATTRIBUTE: data
DATA STORE LOCATION: RUN_PARAMETERS
ACCURACY: N/A

PART II. CONTENTS OF DATA STORES

Table 7.1: DATA STORE: GUIDANCE_STATE

VARIABLE NAME	USED BY:
A_STATUS	2.3 ASP, 2.4 CP
AE_STATUS	2.1 AECLP, 2.4 CP
AE_SWITCH	2.1 AECLP, 2.7 GP
AE_TEMP	2.1 AECLP, 2.4 CP, 2.5 CRCP, 2.7 GP
AR_STATUS	2.2 ARSP, 2.4 CP
C_STATUS	2.4 CP
CHUTE_RELEASED	2.1 AECLP, 2.4 CP, 2.5 CRCP, 2.7 GP
CONTOUR_CROSSED	2.1 AECLP, 2.4 CP, 2.7 GP
FRAME_BEAM_UNLOCKED	2.9 TDLRSP
FRAME_ENGINES_IGNITED	2.1 AECLP, 2.7 GP
G_STATUS	2.4 CP, 2.6 GSP
GP_ALTITUDE	2.4 CP, 2.7 GP, 2.1 AECLP
GP_ATTITUDE	2.4 CP, 2.7 GP
GP_PHASE	2.4 CP, 2.7 GP
GP_ROTATION	2.1 AECLP, 2.4 CP, 2.7 GP
GP_VELOCITY	2.1 AECLP, 2.4 CP, 2.7 GP
INTERNAL_CMD	2.1 AECLP
K_ALT	2.2 ARSP, 2.4 CP, 2.7 GP
K_MATRIX	2.4 CP, 2.7 GP, 2.9 TDLRSP
PE_INTEGRAL	2.1 AECLP, 2.4 CP
RE_STATUS	2.4 CP, 2.8 RECLP
RE_SWITCH	INIT_GCS, 2.7 GP, 2.8 RECLP
TDLR_STATE	2.4 CP, 2.7 GP, 2.9 TDLRSP
TDLR_STATUS	2.4 CP, 2.9 TDLRSP
TDLRSP_SWITCH	INIT_GCS
TDS_STATUS	2.4 CP, 2.7 GP, 2.10 TDSP
TDSP_SWITCH	0. GCS
TE_INTEGRAL	2.1 AECLP, 2.4 CP
TE_LIMIT	2.1 AECLP
THETA	2.8 RECLP
TS_STATUS	2.4 CP, 2.11 TSP
VELOCITY_ERROR	2.1 AECLP, 2.4 CP, 2.7 GP
YE_INTEGRAL	2.1 AECLP, 2.4 CP

Table 7.2: DATA STORE: EXTERNAL

VARIABLE NAME	USED BY
A_COUNTER	2.3 ASP
AE_CMD	2.1 AECLP,2.4 CP
AR_COUNTER	2.2 ARSP
FRAME_COUNTER	2.1 AECLP, 2.4 CP, 2.7 GP, 2.9 TDLRSP
G_COUNTER	2.6 GSP
PACKET	2.4 CP
RE_CMD	2.8 RECLP, 2.4 CP
SS_TEMP	2.11 TSP
TD_COUNTER	2.10 TDSP
TDLR_COUNTER	2.9 TDLRSP
THERMO_TEMP	2.11 TSP

Table 7.3: DATA STORE: SENSOR_OUTPUT

VARIABLE NAME	USED BY:
A_ACCELERATION	2.1 AECLP, 2.3 ASP, 2.4 CP, 2.7 GP
AR_ALTITUDE	2.2 ARSP, 2.4 CP, 2.7 GP
ATMOSPHERIC_TEMP	2.3 ASP, 2.4 CP, 2.6 GSP, 2.11 TSP
G_ROTATION	2.4 CP, 2.6 GSP, 2.7 GP, 2.8 RECLP
TD_SENSED	2.4 CP, 2.7 GP, 2.10 TDSP
TDLR_VELOCITY	2.4 CP, 2.7 GP, 2.9 TDLRSP

Table 7.4: DATA STORE: RUN_PARAMETERS

VARIABLE NAME	USED BY
A_BIAS	2.3 ASP
A_GAIN_0	2.3 ASP
A_SCALE	2.3 ASP
ALPHA_MATRIX	2.3 ASP
AR_FREQUENCY	2.2 ARSP
COMM_SYNC_PATTERN	2.4 CP
CONTOUR_ALTITUDE	2.7 GP
CONTOUR_VELOCITY	2.7 GP
DELTA_T	2.7 GP, 2.8 RECLP, 2.9 TDLRSP
DROP_HEIGHT	2.7 GP
ENGINES_ON_ALTITUDE	2.1 AECLP, 2.7 GP
FULL_UP_TIME	2.1 AECLP
G1	2.3 ASP
G2	2.3 ASP
G3	2.6 GSP
G4	2.6 GSP
G_GAIN_0	2.6 GSP
G_OFFSET	2.6 GSP
GA	2.1 AECLP
GAX	2.1 AECLP
GP1	2.1 AECLP
GP2	2.1 AECLP
GPY	2.1 AECLP
GQ	2.1 AECLP
GR	2.1 AECLP
GRAVITY	2.7 GP
GV	2.1 AECLP
GVE	2.1 AECLP
GVEI	2.1 AECLP
GVI	2.1 AECLP
GW	2.1 AECLP
GWI	2.1 AECLP
M1	2.11 TSP
M2	2.11 TSP
M3	2.11 TSP
M4	2.11 TSP
OMEGA	2.1 AECLP

Table 7.5: DATA STORE: RUN_PARAMETERS (cont.)

VARIABLE NAME	USED BY
P1	2.8 RECLP
P2	2.8 RECLP
P3	2.8 RECLP
P4	2.8 RECLP
PE_MAX	2.1 AECLP
PE_MIN	2.1 AECLP
T1	2.11 TSP
T2	2.11 TSP
T3	2.11 TSP
T4	2.11 TSP
TDLR_ANGLES	2.9 TDLRSP
TDLR_GAIN	2.9 TDLRSP
TDLR_LOCK_TIME	2.9 TDLRSP
TDLR_OFFSET	2.9 TDLRSP
TE_DROP	2.1 AECLP
TE_INIT	2.1 AECLP
TE_MAX	2.1 AECLP
TE_MIN	2.1 AECLP
THETA1	2.8 RECLP
THETA2	2.8 RECLP
YE_MAX	2.1 AECLP
YE_MIN	2.1 AECLP

PART III. LIST OF CONTROL VARIABLES AND DATA CONDITIONS

Table 7.6 CONTROL VARIABLES (OPTIONAL USAGE)

CONTROL VARIABLE NAME
AECLP_DONE
ARSP_DONE
ASP_DONE
CRCP_DONE
GP_DONE
GSP_DONE
TDLRSP_DONE
TDSP_DONE
TSP_DONE

Table 7.7 DATA CONDITIONS (REQUIRED USAGE)

DATA CONDITION VARIABLE NAME
AE_TEMP
CHUTE_RELEASED
TD_SENSED
TDLR_STATE

Table 7.8: INITIALIZATION DATA

VARIABLE NAME	USED BY
A_ACCELERATION	2.1 AECLP, 2.3 ASP, 2.4 CP, 2.7 GP
A_BIAS	2.3 ASP
A_COUNTER	2.3 ASP
A_GAIN_0	2.3 ASP
A_SCALE	2.3 ASP
A_STATUS	2.3 ASP, 2.4 CP
AE_STATUS	2.1 AECLP, 2.4 CP
AE_SWITCH	2.1 AECLP, 2.7 GP
AE_TEMP	2.1 AECLP, 2.4 CP, 2.5 CRCP, 2.7 GP
ALPHA_MATRIX	2.3 ASP
AR_ALTITUDE	2.2 ARSP, 2.4 CP, 2.7 GP
AR_COUNTER	2.2 ARSP
AR_FREQUENCY	2.2 ARSP
AR_STATUS	2.2 ARSP, 2.4 CP
ATMOSPHERIC_TEMP	2.3 ASP, 2.4 CP, 2.6 GSP, 2.11 TSP
C_STATUS	2.4 CP
CHUTE_RELEASED	2.1 AECLP, 2.4 CP, 2.5 CRCP, 2.7 GP
COMM_SYNC_PATTERN	2.4 CP
CONTOUR_ALTITUDE	2.7 GP
CONTOUR_CROSSED	2.1 AECLP, 2.4 CP, 2.7 GP
CONTOUR_VELOCITY	2.7 GP
DELTA_T	2.7 GP
DROP_HEIGHT	2.7 GP
ENGINES_ON_ALTITUDE	2.1 AECLP, 2.7 GP
FRAME_BEAM_UNLOCKED	2.9 TDLRSP
FRAME_COUNTER	2.1 AECLP, 2.4 CP, 2.7 GP, 2.9 TDLRSP
FRAME_ENGINES_IGNITED	2.1 AECLP, 2.7 GP
FULL_UP_TIME	2.1 AECLP

Table 7.9: INITIALIZATION DATA (cont.)

VARIABLE NAME	USED BY
G1	2.3 ASP
G2	2.3 ASP
G3	2.6 GSP
G4	2.6 GSP
G_COUNTER	2.6 GSP
G_GAIN_0	2.6 GSP
G_OFFSET	2.6 GSP
G_ROTATION	2.4 CP, 2.6 GSP, 2.7 GP, 2.8 RECLP
G_STATUS	2.4 CP, 2.6 GSP
GA	2.1 AECLP
GAX	2.1 AECLP
GP1	2.1 AECLP
GP2	2.1 AECLP
GP_ALTITUDE	2.7 GP, 2.1 AECLP
GP_ATTITUDE	2.7 GP
GP_PHASE	2.4 CP, 2.7 GP
GP_ROTATION	2.7 GP, 2.8 RECLP
GP_VELOCITY	2.7 GP
GPY	2.1 AECLP
GQ	2.1 AECLP
GR	2.1 AECLP
GRAVITY	2.7 GP
GV	2.1 AECLP
GVE	2.1 AECLP
GVEI	2.1 AECLP
GVI	2.1 AECLP
GW	2.1 AECLP
GWI	2.1 AECLP

C-2

Table 7.10: INITIALIZATION DATA (cont.)

VARIABLE NAME	USED BY
K_ALT	2.2 ARSP, 2.4 CP, 2.7 GP
K_MATRIX	2.4 CP, 2.7 GP, 2.9 TDLRSP
M1	2.11 TSP
M2	2.11 TSP
M3	2.11 TSP
M4	2.11 TSP
OMEGA	2.1 AECLP
P1	2.8 RECLP
P2	2.8 RECLP
P3	2.8 RECLP
P4	2.8 RECLP
PE_INTEGRAL	2.1 AECLP, 2.4 CP
PE_MAX	2.1 AECLP
PE_MIN	2.1 AECLP
RE_STATUS	2.4 CP, 2.8 RECLP
RE_SWITCH	INIT_GCS, 2.7 GP, 2.8 RECLP
SS_TEMP	2.11 TSP
T1	2.11 TSP
T2	2.11 TSP
T3	2.11 TSP
T4	2.11 TSP
TD_SENSED	2.4 CP, 2.7 GP, 2.10 TDSP
TDLR_ANGLES	2.9 TDLRSP
TDLR_COUNTER	2.10 TDSP
TDLR_GAIN	2.9 TDLRSP
TDLR_LOCK_TIME	2.9 TDLRSP
TDLR_OFFSET	2.9 TDLRSP
TDLR_STATE	2.4 CP, 2.7 GP, 2.9 TDLRSP
TDLR_STATUS	2.4 CP, 2.9 TDLRSP
TDLR_VELOCITY	2.4 CP, 2.7 GP, 2.9 TDLRSP

Table 7.11: INITIALIZATION DATA (cont.)

VARIABLE NAME	USED BY
TDLRSP_SWITCH	INIT_GCS
TDS_STATUS	2.4 CP, 2.7 GP, 2.10 TDSP
TDSP_SWITCH	0. GCS
TE_DROP	2.1 AECLP
TE_INIT	2.1 AECLP
TE_INTEGRAL	2.1 AECLP, 2.4 CP
TE_LIMIT	2.1 AECLP
TE_MAX	2.1 AECLP
TE_MIN	2.1 AECLP
THERMO_TEMP	2.11 TSP
THETA	2.8 RECLP
THETA1	2.8 RECLP
THETA2	2.8 RECLP
TS_STATUS	2.4 CP, 2.11 TSP
VELOCITY_ERROR	2.1 AECLP, 2.4 CP, 2.7 GP
YE_INTEGRAL	2.1 AECLP, 2.4 CP
YE_MAX	2.1 AECLP
YE_MIN	2.1 AECLP

A. FORMAT OF THIS SPECIFICATION

INTRODUCTION TO FORMAT

This specification uses the extended structured analysis method advocated by Hatley [12, 13]. 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, control, and timing descriptions; and a requirements dictionary. Figure A.1 defines the graphical symbols used in the data flow and control flow diagrams respectively.

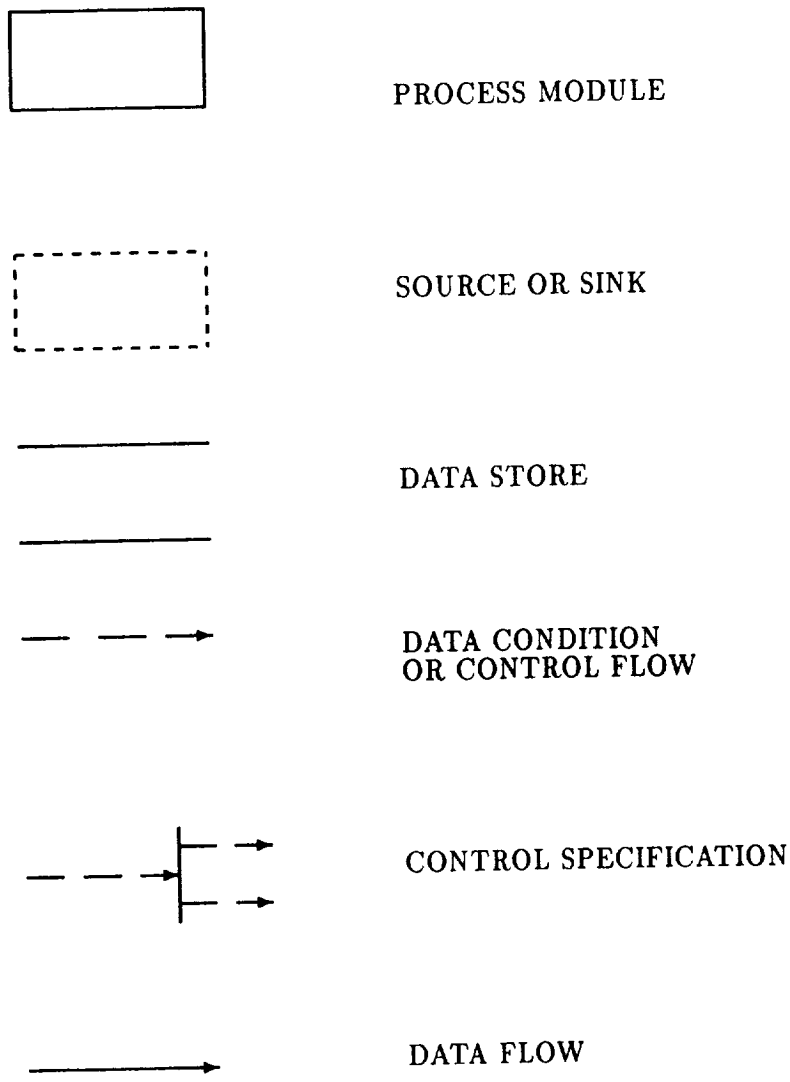
The data flow diagrams describe the processes, data flows, data stores, and data conditions. The data context diagram is the highest-level data flow diagram and represents the data flow for the entire system. Data conditions are represented by directed arcs with broken lines.

The control flow diagrams describe processes, control signal flows, and stores. The control signal 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. They 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. This duplication of processes is consistent with the approach of slaving the control flow to the data flow. 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 requirements dictionary contains definitions for both data and control signals.

Figure A.1: GRAPHICAL SYMBOLS USED IN FLOW DIAGRAMS



B. IMPLEMENTATION NOTES

INTERFACE

Background

For the purposes of this research experiment, each GCS software 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.

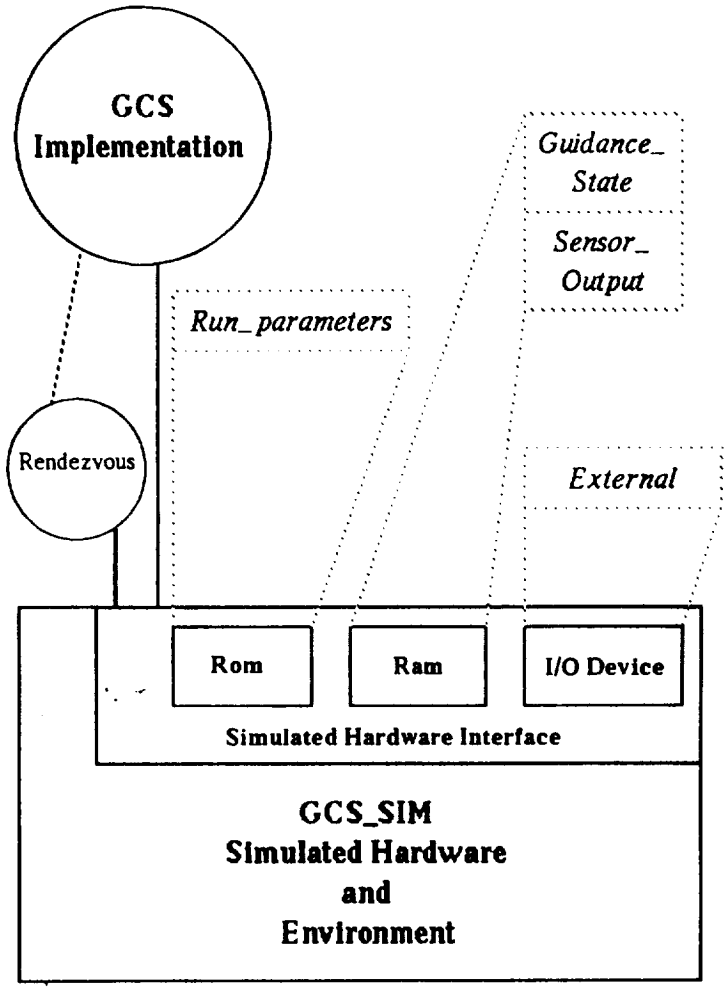
Simulator Support Utility

A single simulator support utility (GCS_SIM_RENDEZVOUS) is provided to form a uniform interface between the GCS applications software 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

Global Data Store Organization

Part III of the data dictionary of this specification contains descriptions of four required data stores. Each of these data stores is to be located in a separate, globally accessible data region. By dividing the global data stores into four separate regions, they can be compared to components that would be found on a hardware interface (See Figure B.1).



Hardware Component	Global Data Store (Software Interface)
Input/Output Device	EXTERNAL
Read-Only Memory	RUN_PARAMETERS
On-board Random Access Memory	GUIDANCE_STATE
On-board Random Access Memory	SENSOR_OUTPUT

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

In FORTRAN, this would mean four common blocks with the labels as given in the header for each data store listing. There are ways of accomplishing this same type of data region in other languages, but they are outside the scope of this experiment.

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 application will be transferred through the four global data stores defined in the data dictionary. The programmer should note that although normally some type of range checking/limiting would be included in control programs, there are some restrictions being placed by this experiment. The programmer is allowed to *check* values of variables to see if they are within the ranges specified in the data dictionary, but if values are outside the specified range, **NO CHANGES** should be made to them. For purposes of this experiment, the calculated values need to be passed to the simulator. Values returned from the simulator will always be within the specified range, so if the application sends out-of-range values to the simulator, these values will be put into range before being passed from the simulator to the next subframe processes. This means that all inputs to subframes may be assumed to be within the specified ranges.

Process

The GCS *reads* the sensor input values and processes them into control commands which are averaged by GCS_SIM and written to the control surfaces. 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 application. Following the first call to rendezvous (time step equal to zero), all GCS applications will synchronize themselves by calling rendezvous at the **end** of each sub-frame. This rendezvous, in effect, suspends the GCS implementations until the other GCS implementations have processed this time step or have run out of time.

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 rendezvous - the frame counter value will be updated with the starting frame for the particular trajectory. Under *normal* circumstances, the value of the frame counter will be "1," but **do not** rely on that. As errors occur in the GCS, they will be fixed; the trajectory may start at the beginning of the last complete frame that was processed without error.

Local Variables

In an attempt to accommodate everyone, most of the variables needed to manipulate functions within the GCS have been included in the data stores, which can be found in the data dictionary. Since a GCS can be started at the beginning of any frame, the programmer is responsible for establishing acceptable initialization values for any local variables (any variable not listed in the data dictionary) which may have been declared. Assume that some of the GCS_SIM may initialize the GCS with a list of variables from some saveset of previous global data store values.

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

C. NUMERICAL INTEGRATION INSTRUCTIONS

Three locations exist within the GCS specification requiring the use of a highly accurate numerical integration method[1]. These locations are the calculations of GP_VELOCITY, GP_ALTITUDE, and GP_ATTITUDE in the Guidance Processor. To maintain the necessary degree of accuracy in certain output variables, three methods of numerical integration have been designated as acceptable for coding: Adams-Moulton method, Hamming's method, and the Runge-Kutta fourth-order method.

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 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 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 to 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 The well-known Runge-Kutta fourth-order method requires only the previous two values to calculate the next value. References can be found in many texts including; [14] (pp. 352-8), [15] (pp. 273-80), [16] (pp. 481-6), and [17] (pp. 152-4).

[1]Note: not all integration required by the GCS specification requires the use of one of the methods listed in this appendix. More specifically, in computing 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).

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)

To insure that the numerical integration scheme coded provides sufficient accuracy in the output variable, an Accuracy Validation Utility Program (AVUP) will be used during acceptance testing.

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 of 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 & Row, New York, 1979.

[15] J. B. Scarborough. *Numerical Mathematical Analysis.* The Johns Hopkins Press, Baltimore, 1930.

[16] Stephen M. Pizer. *Numerical Computing and Mathematical Analysis.* Science Research Associates, Inc., Chicago, 1975.

[17] Philip J. Davis and Philip Rabinowitz. *Methods of Numerical Integration.* Academic Press, New York, 1975.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-182058		

4. Title and Subtitle	5. Report Date
Software Requirements: Guidance and Control Software Development Specification	June 1990
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
B. E. Withers, D. C. Rich, D. S. Lowman, and R. C. Buckland	
	10. Work Unit No. 505-66-21-01
9. Performing Organization Name and Address Research Triangle Institute P.O. Box 12194 Research Triangle Park, NC 27709-2194	11. Contract or Grant No. NAS1-17964
	13. Type of Report and Period Covered Contractor Report
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Langley Research Center Hampton, VA 23665-5225	14. Sponsoring Agency Code

15. Supplementary Notes

Technical Monitor: George B. Finelli, Langley Research Center
Task 8 Report

16. Abstract

This document specifies the software requirements for an implementation of Guidance and Control Software (GCS). The purpose of the GCS is to provide guidance and engine control to a planetary landing vehicle during its terminal descent onto a planetary surface and to communicate sensory information about that vehicle and its descent to some receiving device. The specification was developed using the structured analysis for real-time system specification methodology by Hatley and Pirbhai and was based on a simulation program used to study the probability of success of the 1976 Viking Lander missions to Mars. Three versions of GCS are being generated for use in software error studies research conducted by the Research Triangle Institute and the NASA Langley Research Center.

17. Key Words (Suggested by Author(s))	18. Distribution Statement
Software Requirements Specification Guidance and Control Software (GCS) Planetary Landing Vehicle	Unclassified-Unlimited Subject Category 61

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of pages	22. Price
Unclassified	Unclassified	137	A07