AD NUMBER

ADB120255

LIMITATION CHANGES

TO:

Approved for public release; distribution is unlimited.

FROM:

Distribution authorized to U.S. Gov't. agencies and their contractors; Critical Technology; 30 OCT 1987. Other requests shall be referred to Air Force Armament Lab., Eglin AFB, FL 32542. This document contains export-controlled technical data.

AUTHORITY

AFSC/MNOI wright lab from Eglin AFB, ltr dtd 13 Feb 1992

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

DTIC FILE COPY

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Distribution authorized to U.S. Government Agencies and their contractors; CT (over)
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFATL-TR-88-18, Vol 8
6a. NAME OF PERFORMING ORGANIZATION McDonnell Douglas Astronautics Company 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Aeromechanics Division
6c. ADDRESS (City, State, and ZIP Code) P.O. Box 516 St. Louis, MO 63166	7b. ADDRESS (City, State, and ZIP Code) Air Force Armament Laboratory Eglin AFB, FL 32542-5434
8a. NAME OF FUNDING / SPONSORING ORGANIZATION STARS Joint Program Office 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F08635-86-C-0025
8c. ADDRESS (City, State, and ZIP Code) Room 3D139 (1211 Fern St) The Pentagon Washington DC 20301-3081	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. 63756D; PROJECT NO. 921C; TASK NO. GZ; WORK UNIT ACCESSION NO. 57

11. TITLE (Include Security Classification)
Common Ada Missile Package (CAMP) Project: Missile Software Parts, Vol 8: Detail Design Documents (Vol 7-12)

12. PERSONAL AUTHOR(S)
D. McNicholl, S. Cohen, C. Palmer, et al.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical Note	FROM Sep 85 TO Mar 88	March 1988	282

16. SUPPLEMENTARY NOTATION
SUBJECT TO EXPORT CONTROL LAWS.
Availability of this report is specified on verso of front cover. (over)

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Reusable Software, Missile Software, Software Generators
Ada, Parts Composition Systems, Software Parts

19. ABSTRACT (Continue on reverse if necessary and identify by block number)
The objective of the CAMP program is to demonstrate the feasibility of reusable Ada software parts in a real-time embedded application area; the domain chosen for the demonstration was that of missile flight software systems. This required that the existence of commonality within that domain be verified (in order to justify the development of parts for that domain), and that software parts be designed which address those areas identified. An associated parts system was developed to support parts usage. Volume 1 of this document is the User's Guide to the CAMP Software parts; Volume 2 is the Version Description Document; Volume 3 is the Software Product Specification; Volumes 4-6 contain the Top-Level Design Document; and, Volumes 7-12 contain the Detail Design Documents.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Christine Anderson	22b. TELEPHONE (Include Area Code) (904) 882-2961
	22c. OFFICE SYMBOL AFATL/FXG

DD Form 1473, JUN 86 Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

3. DISTRIBUTION/AVAILABILITY OF REPORT (CONCLUDED)

~~this report documents test and evaluation~~ distribution limitation applied March 1988. Other requests for this document must be referred to AFATL/FXG, Eglin AFB, Florida 32542-5434.

16. SUPPLEMENTARY NOTATION (CONCLUDED)

These technical notes accompany the CAMP final report AFATL-TR-85-93 (3 Vols)

AFATL-TR-88-18, Vol 8

SOFTWARE DETAILED DESIGN DOCUMENT

FOR THE

MISSILE SOFTWARE PARTS

OF THE

COMMON ADA MISSILE PACKAGE (CAMP)
PROJECT

CONTRACT F08635-86-C-0025

CDRL SEQUENCE NO. C007

30 OCTOBER 1987

AIR FORCE ARMAMENT LABORATORY
Air Force Systems Command ▮ United States Air Force ▮ Eglin Air Force Base, Florida

3.3.4 GUIDANCE AND CONTROL

(This page intentionally left blank.)

3.3.4.1 WAYPOINT_STEERING (PACKAGE BODY) TLCSC P661 (CATALOG #P106-0)

This package contains the CAMP parts required to do the waypoint steering portion of navigation.

The following three waypoints are required to perform waypoint steering: o A : the last waypoint passed by the missile o B : the waypoint to which the missile is currently heading o C : the next waypoint to which the missile will head

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.4.1.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
Steering_Vector_Operations	R170, R171
Steering_Vector_Operations_with_Arcsin	N/A
Compute_Turn_Angle_and_Direction	R172
Crosstrack_And_Heading_Error_Operations	R173, R174, R175
Distance_to_Current_Waypoint	R176
Distance_to_Current_Waypoint_with_Arcsin	N/A
Compute_Turning_and_Nonturning_Distances	R177
Turn_Test_Operations	R178, R179, R180

3.3.4.1.2 LOCAL ENTITIES DESIGN

None.

3.3.4.1.3 INPUT/OUTPUT

None.

3.3.4.1.4 LOCAL DATA

None.

3.3.4.1.5 PROCESS CONTROL

Not applicable.

3.3.4.1.6 PROCESSING

The following describes the processing performed by this part:

```
package body Waypoint_Steering is

   package body Steering_Vector_Operations is separate;

   package body Steering_Vector_Operations_with_Arcsin is separate;

   procedure Compute_Turn_Angle_and_Direction
               (Unit_Normal_C              : in     Unit_Vectors;
                Unit_Tangent_B             : in     Unit_Vectors;
                Unit_Tangent_C             : in     Unit_Vectors;
                Tan_of_One_Half_Turn_Angle :    out Tan_Ratio;
                Turn_Direction             :    out Turning_Directions)
               is separate;

   package body Crosstrack_and_Heading_Error_Operations is separate;

   function Distance_to_Current_Waypoint
              (Unit_Radial_M  : Unit_Vectors;
               Unit_Tangent_B : Unit_Vectors)
              return Segment_Distances is separate;

   function Distance_to_Current_Waypoint_with_Arcsin
              (Unit_Radial_M  : Unit_Vectors;
               Unit_Tangent_B : Unit_Vectors)
              return Segment_Distances is separate;

   procedure Compute_Turning_and_Nonturning_Distances
               (Tan_of_One_Half_Turn_Angle : in     Tan_Ratio;
                Segment_BC_Distance        : in     Distances;
                Turn_Radius                : in     Distances;
                Turning_Distance           :    out Distances;
                Nonturning_Distance        :    out Distances) is separate;

   package body Turn_Test_Operations is separate;

end Waypoint_Steering;
```

3.3.4.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.4.1.8 LIMITATIONS

None.

3.3.4.1.9 LLCSC DESIGN

3.3.4.1.9.1 STEERING_VECTOR_OPERATIONS (PACKAGE BODY) PACKAGE DESIGN (CATALOG #P107-0)

This package contains operations to do the following: o Initialize the waypoint steering vectors when supplied with the latitude and longitude of the past, current, and next waypoints o Update the waypoint steering vectors when supplied with the latitude and longitude of the "new" waypoint, C.

The waypoint steering vectors for a course segment, extending from waypoint A to waypoint B, are the segment unit normal vector (UN_B) and the segment unit tangent vector (UT_B).

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.4.1.9.1.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
Initialize	R170
Update	R171

3.3.4.1.9.1.2 LOCAL ENTITIES DESIGN

None.

3.3.4.1.9.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined with the specification for this part.

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Indices	discrete type	Used to dimension Unit_Vectors
Earth_Distances	floating point type	Data type used to define distance measurements of the Earth's radius
Segment_Distances	floating point type	Data type used to define distance measurements of the navigation segments
Earth_Positions	floating point type	Data type used to define latitude and longitude measurements
Sin_Cos_Ratio	floating point type	Data type used to define results of a sine or cosine function
Unit_Vectors	array	Array of "Sin_Cos_Ratio" dimensioned by Indices

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Description
Earth_Radius	Earth_Distances	Radius of the Earth

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Operator defining the operation: Earth_Distances * Sin_Cos_Ratio => Segment_Distances
"/"	function	Operator defining the operation: Unit_Vectors / Sin_Cos_Ratio => Unit_Vectors
Cross_Product	procedure	Cross product function
Vector_Length	function	Calculates the length of a vector
Sin_Cos	procedure	Calculates the sine and cosine of an input value

3.3.4.1.9.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Unit_Radial_B	Unit_Vectors	N/A	Unit radial vector to waypoint B extending outwards from the origin of the Earth-centered reference frame towards waypoint B
Unit_Radial_C	Unit_Vectors	N/A	Unit radial vector to waypoint C extending outwards from the origin of the Earth-centered reference frame towards waypoint C

3.3.4.1.9.1.5 PROCESS CONTROL

Not applicable.

3.3.4.1.9.1.6 PROCESSING

The following describes the processing performed by this part:

```
with Geometric_Operations;
separate (Waypoint_Steering)
package body Steering_Vector_Operations is

   package Geo renames Geometric_Operations;

-- ------------------------------
-- --instantiate required parts-
-- ------------------------------

   function U_R_Vector is new
               Geo.Unit_Radial_Vector (Indices          => Indices,
                                       Earth_Positions => Earth_Positions,
                                       Sin_Cos_Ratio    => Sin_Cos_Ratio,
                                       Unit_Vectors     => Unit_Vectors);

   procedure Compute_Segment_and_U_Nl_Vector is new
                Geo.Compute_Segment_and_Unit_Normal_Vector
                   (Indices             => Indices,
                    Earth_Distances     => Earth_Distances,
                    Segment_Distances   => Segment_Distances,
                    Sin_Cos_Ratio       => Sin_Cos_Ratio,
                    Unit_Vectors        => Unit_Vectors,
                    Earth_Radius        => Earth_Radius);

-- ---------------------
-- --local declarations-
-- ---------------------

   Unit_Radial_B : Unit_Vectors;
   Unit_Radial_C : Unit_Vectors;
```

```
-- ----------------------
-- --separate procedures-
-- ----------------------

   procedure Initialize
               (Waypoint_A_Lat      : in  Earth_Positions;
                Waypoint_A_Long     : in  Earth_Positions;
                Waypoint_B_Lat      : in  Earth_Positions;
                Waypoint_B_Long     : in  Earth_Positions;
                Waypoint_C_Lat      : in  Earth_Positions;
                Waypoint_C_Long     : in  Earth_Positions;
                Unit_Normal_B       : out Unit_Vectors;
                Unit_Normal_C       : out Unit_Vectors;
                Unit_Tangent_B      : out Unit_Vectors;
                Unit_Tangent_C      : out Unit_Vectors;
                Segment_BC_Distance : out Segment_Distances) is separate;

   procedure Update
               (Waypoint_C_Lat      : in      Earth_Positions;
                Waypoint_C_Long     : in      Earth_Positions;
                Unit_Normal_B       :     out Unit_Vectors;
                Unit_Normal_C       : in  out Unit_Vectors;
                Unit_Tangent_B      :     out Unit_Vectors;
                Unit_Tangent_C      : in  out Unit_Vectors;
                Segment_BC_Distance :     out Segment_Distances) is separate;

end Steering_Vector_Operations;
```

3.3.4.1.9.1.7 UTILIZATION OF OTHER ELEMENTS

The following library units are with'd by this part:
1. Geometric_Operations package (Geo)

UTILIZATION OF EXTERNAL ELEMENTS:

Subprograms and task entries:

The following table summarizes the external subroutines and task entries required by this part:

Name	Type	Source	Description
Unit_Radial_Vector	generic function	Geo	Computes a unit radial vector
Compute_Segment_and_Unit_Normal_Vector	generic function	Geo	Computes segment distance and unit normal vectir

3.3.4.1.9.1.8 LIMITATIONS

None.

3.3.4.1.9.1.9 LLCSC DESIGN

None.

3.3.4.1.9.1.10 UNIT DESIGN

3.3.4.1.9.1.10.1 INITIALIZE (PROCEDURE BODY) UNIT DESIGN

This part initializes the waypoint steering vectors when supplied with the latitude and longitude of the past (A), present (B), and next (C) waypoints.

This part initializes the waypoint steering vectors for the "current" course segment AB, as well as for the "next" course segment BC.

3.3.4.1.9.1.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R170.

3.3.4.1.9.1.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.4.1.9.1.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Waypoint_A_Lat, Waypoint_A_Long	Earth_Positions	in	Latitude and longitude of the "previous" waypoint
Waypoint_B_Lat, Waypoint_B_Long	Earth_Positions	in	Latitude and longitude of the "current" waypoint
Waypoint_C_Lat, Waypoint_C_Long	Earth_Positions	in	Latitude and longitude of the "next" waypoint
Unit_Normal_B, Unit_Normal_C	Unit_Vectors	out	Unit normal vectors for segments AB and BC
Unit_Tangent_B, Unit_Tangent_C	Unit_Vectors	out	Unit tangent vectors for segments AB and BC
Segment_BC_ Distance	Segment_ Distances	out	Great circle arclength between waypoints B and C

3.3.4.1.9.1.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Temp_UN_B	Unit_Vectors	N/A	Temporary Unit_Normal_B vector
Temp_UN_C	Unit_Vectors	N/A	Temporary Unit_Normal_C vector
Unit_Radial_A	Unit_Vectors	N/A	Unit radial vector pointing to waypoint A
V_Length	Sin_Cos_Ratio	N/A	Vector length

3.3.4.1.9.1.10.1.5 PROCESS CONTROL

Not applicable.

3.3.4.1.9.1.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Waypoint_Steering.Steering_Vector_Operations)
procedure Initialize
             (Waypoint_A_Lat      : in  Earth_Positions;
              Waypoint_A_Long     : in  Earth_Positions;
              Waypoint_B_Lat      : in  Earth_Positions;
              Waypoint_B_Long     : in  Earth_Positions;
              Waypoint_C_Lat      : in  Earth_Positions;
              Waypoint_C_Long     : in  Earth_Positions;
              Unit_Normal_B       : out Unit_Vectors;
              Unit_Normal_C       : out Unit_Vectors;
              Unit_Tangent_B      : out Unit_Vectors;
              Unit_Tangent_C      : out Unit_Vectors;
              Segment_BC_Distance : out Segment_Distances) is

-- ----------------------
-- --declaration section-
-- ----------------------

   Temp_UN_B     : Unit_Vectors;
   Temp_UN_C     : Unit_Vectors;
   Unit_Radial_A : Unit_Vectors;
   V_Length      : Sin_Cos_Ratio;

-----------------------------
--begin procedure Initialize-
-----------------------------

begin

-- --compute unit radial vectors

   Unit_Radial_A := U_R_Vector(Lat_of_Point  => Waypoint_A_Lat,
                               Long_of_Point => Waypoint_A_Long);
   Unit_Radial_B := U_R_Vector(Lat_of_Point  => Waypoint_B_Lat,
                               Long_of_Point => Waypoint_B_Long);
   Unit_Radial_C := U_R_Vector(Lat_of_Point  => Waypoint_C_Lat,
                               Long_of_Point => Waypoint_C_Long);
```

```
-- --compute UN_B

   Temp_UN_B      := Cross_Product(Left  => Unit_Radial_B,
                                  Right => Unit_Radial_A);
   V_Length       := Vector_Length(Temp_UN_B);
   Temp_UN_B      := Temp_UN_B / V_Length;
   Unit_Normal_B  := Temp_UN_B;

-- --compute UT_B

   Unit_Tangent_B := Cross_Product(Left  => Temp_UN_B,
                                  Right => Unit_Radial_B);

-- --compute UN_C and segment BC distance

   Compute_Segment_and_U_Nl_Vector
      (Unit_Radial1     => Unit_Radial_B,
       Unit_Radial2     => Unit_Radial_C,
       Unit_Normal2     => Temp_UN_C,
       Segment_Distance => Segment_BC_Distance);

   Unit_Normal_C  := Temp_UN_C;

-- --compute UT_C

   Unit_Tangent_C := Cross_Product(Left  => Temp_UN_C,
                                  Right => Unit_Radial_C);

end Initialize;
```

3.3.4.1.9.1.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and instantiated in the body of the Steering_Vector_Operations package:

Name	Type	Description
U_R_Vector	function	Computes a unit radial vector
Compute_Segment_and_U_Nl_Vector	function	Computes segment distance and unit normal vector

The following table describes the subroutines required by this part and defined as generic formal subroutines to the Steering_Vector_Operations package:

Name	Type	Description
"/"	function	Operator defining the operation: Unit_Vectors / Sin_Cos_Ratio => Unit_Vectors
Cross_Product	procedure	Cross product function
Vector_Length	function	Calculates the length of a vector

Data types:

The following data types are required by this part and defined as generic parameters to the Steering_Vector_Operations package:

Name	Type	Description
Indices	discrete type	Used to dimension Unit_Vectors
Earth_Distances	floating point type	Data type used to define distance measurements of the Earth's radius
Segment_Distances	floating point type	Data type used to define distance measurements of the navigation segments
Earth_Positions	floating point type	Data type used to define latitude and longitude measurements
Sin_Cos_Ratio	floating point type	Data type used to define results of a sine or cosine function
Unit_Vectors	array	Array of "Sin_Cos_Ratio" dimensioned by Indices

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Steering_Vector_Operations:

Name	Type	Value	Description
Unit_Radial_B	Unit_Vectors	N/A	Unit radial vector to waypoint B extending outwards from the origin of the Earth-centered reference frame towards waypoint B
Unit_Radial_C	Unit_Vectors	N/A	Unit radial vector to waypoint C extending outwards from the origin of the Earth-centered reference frame towards waypoint C

3.3.4.1.9.1.10.1.8 LIMITATIONS

None.

3.3.4.1.9.1.10.2 UPDATE (PROCEDURE BODY) UNIT DESIGN

This part updates the waypoint steering vectors when supplied with the latitude and longitude of the "new" waypoint, C.

The waypoint steering vectors for a course segment, extending from waypoint A to waypoint B, are the segment unit normal vector (UN_B) and the segment unit tangent vector (UT_B).

3.3.4.1.9.1.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R171.

3.3.4.1.9.1.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.4.1.9.1.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Waypoint_C_Lat, Waypoint_C_Long	Earth_Positions	in	Latitude and longitude of the "next" waypoint
Unit_Normal_B, Unit_Normal_C	Unit_Vectors	out in out	Unit normal vectors for segments AB and BC
Unit_Tangent_B, Unit_Tangent_C	Unit_Vectors	out in out	Unit tangent vectors for segments AB and BC
Segment_BC_ Distance	Segment_ Distances	out	Great circle arclength between waypoints B and C

3.3.4.1.9.1.10.2.4 LOCAL DATA

None.

3.3.4.1.9.1.10.2.5 PROCESS CONTROL

Not applicable.

3.3.4.1.9.1.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Waypoint_Steering.Steering_Vector_Operations)
procedure Update
             (Waypoint_C_Lat      : in     Earth_Positions;
              Waypoint_C_Long     : in     Earth_Positions;
              Unit_Normal_B       :    out Unit_Vectors;
              Unit_Normal_C       : in out Unit_Vectors;
              Unit_Tangent_B      :    out Unit_Vectors;
              Unit_Tangent_C      : in out Unit_Vectors;
              Segment_BC_Distance :    out Segment_Distances) is
begin

-- --advance "C" vectors into "B" vectors

   Unit_Radial_B  := Unit_Radial_C;
   Unit_Normal_B  := Unit_Normal_C;
   Unit_Tangent_B := Unit_Tangent_C;

-- --calculate new values

   Unit_Radial_C  := U_R_Vector(Lat_of_Point  => Waypoint_C_Lat,
                                Long_of_Point => Waypoint_C_Long);

   Compute_Segment_and_U_Nl_Vector
      (Unit_Radial1     => Unit_Radial_B,
       Unit_Radial2     => Unit_Radial_C,
       Unit_Normal2     => Unit_Normal_C,
       Segment_Distance => Segment_BC_Distance);

   Unit_Tangent_C := Cross_Product(Left  => Unit_Normal_C,
                                   Right => Unit_Radial_C);

end Update;
```

3.3.4.1.9.1.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and instantiated in the body of the Steering_Vector_Operations package:

Name	Type	Description
U_R_Vector	function	Computes a unit radial vector
Compute_Segment_and_U_Nl_Vector	function	Computes segment distance and unit normal vector

The following table describes the subroutines required by this part and defined as generic formal subroutines to the Steering_Vector_Operations package:

Name	Type	Description
Cross_Product	procedure	Cross product function

Data types:

The following data types are required by this part and defined as generic parameters to the Steering_Vector_Operations package:

Name	Type	Description
Indices	discrete type	Used to dimension Unit_Vectors
Earth_Distances	floating point type	Data type used to define distance measurements of the Earth's radius
Segment_Distances	floating point type	Data type used to define distance measurements of the navigation segments
Earth_Positions	floating point type	Data type used to define latitude and longitude measurements
Sin_Cos_Ratio	floating point type	Data type used to define results of a sine or cosine function
Unit_Vectors	array	Array of "Sin_Cos_Ratio" dimensioned by Indices

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Steering_Vector_Operations:

Name	Type	Value	Description
Unit_Radial_B	Unit_Vectors	N/A	Unit radial vector to waypoint B extending outwards from the origin of the Earth-centered reference frame towards waypoint B
Unit_Radial_C	Unit_Vectors	N/A	Unit radial vector to waypoint C extending outwards from the origin of the Earth-centered reference frame towards waypoint C

3.3.4.1.9.1.10.2.8 LIMITATIONS

None.

3.3.4.1.9.2 CROSSTRACK_AND_HEADING_ERROR_OPERATIONS (PACKAGE BODY) PACKAGE DESIGN (CATALOG #P109-0)

This part contains the routines require to compute the crosstrack and heading errors for a missile in turning or nonturning flight.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.4.1.9.2.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
Compute_When_Turning	R173
Compute_When_Not_Turning	R175
Compute	R174

3.3.4.1.9.2.2 LOCAL ENTITIES DESIGN

None.

3.3.4.1.9.2.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were defined when this part was originally specified:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Navigation_Indices	discrete type	Data type used to dimension Velocity_Vectors
Unit_Indices	discrete type	Data type used to dimension Unit_Vectors
Angles	floating point type	Data type of angular measurements
Earth_Distances	floating point type	Data type used to define distance measurements of the Earth's radius
Segment_Distances	floating point type	Data type used to define distance measurements of the navigation segments
Sin_Cos_Ratio	floating point type	Data type of results of sine/cosine operations
Tan_Ratio	floating point type	Data type of tangent operations
Velocities	floating point type	Data type of velocity measurements
Unit_Vectors	array	Array, dimensioned by Unit_Indices, of Sin_Cos_Ratio
Velocity_Vectors	array	Array, dimensioned by Navigation_Indices, of Velocities

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
E(ast)	Navigation_Indices	'FIRST	Used to access first element of arrays dimensioned by Navigation_Indices
N(orth)	Navigation_Indices	'SUCC(E)	Used to access second element of arrays dimensioned by Navigation_Indices
U(p)	Navigation_Indices	'LAST	Used to access last element of arrays dimensioned by Navigation_Indices
X	Unit_Indices Indices	'FIRST	Used to access first element of arrays dimensioned by Unit_Indices
Y	Unit_Indices Indices	'SUCC(X)	Used to access second element of arrays dimensioned by Unit_Indices
Z	Unit_Indices Indices	'LAST	Used to access last element of arrays dimensioned by Unit_Indices
Earth_Radius	Earth_Distances	n/a	Radius of the Earth

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplication operator defining the operation: Sin_Cos_Ratio * Earth_Distances => Segment_Distances
"*"	function	Multiplication operator defining the operation: Sin_Cos_Ratio * Segment_Distances => Segment_Distances
"*"	function	Multiplication operator defining the operation: Segment_Distances * INTEGER => Segment_Distances
"*"	function	Multiplication operator defining the operation: INTEGER * Sin_Cos_Ratio => Sin_Cos_Ratio
"*"	function	Multiplication operator defining the operation: Segment_Distances * Velocities => Tan_Ratio
"*"	function	Multiplication operator defining the operation: Sin_Cos_Ratio * Velocities => Velocities
"/"	function	Division operator defining the operation: Velocities / Velocities => Tan_Ratio
Dot_Product	function	Dot product function
Sqrt	function	Square root function
Arctan	function	Arctangent function

3.3.4.1.9.2.4 LOCAL DATA

None.

3.3.4.1.9.2.5 PROCESS CONTROL

Not applicable.

3.3.4.1.9.2.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Waypoint_Steering)
package body Crosstrack_and_Heading_Error_Operations is

   procedure Compute_When_Turning
               (Distance_to_B       : in  Segment_Distances;
                Nonturning_Distance : in  Segment_Distances;
                Unit_Radial_M       : in  Unit_Vectors;
                Unit_Normal_B       : in  Unit_Vectors;
                Unit_Tangent_B      : in  Unit_Vectors;
                Turn_Direction      : in  Turning_Directions;
                Ground_Velocity     : in  Velocity_Vectors;
                Turn_Radius         : in  Segment_Distances;
```

```
                Crosstrack_Error    : out Segment_Distances;
                Heading_Error       : out Angles) is separate;

   procedure Compute_When_Not_Turning
               (Unit_Radial_M     : in  Unit_Vectors;
                Unit_Normal_B     : in  Unit_Vectors;
                Ground_Velocity   : in  Velocity_Vectors;
                Crosstrack_Error  : out Segment_Distances;
                Heading_Error     : out Angles) is separate;

   procedure Compute
               (Distance_to_B       : in  Segment_Distances;
                Nonturning_Distance : in  Segment_Distances;
                Unit_Radial_M       : in  Unit_Vectors;
                Unit_Normal_B       : in  Unit_Vectors;
                Unit_Tangent_B      : in  Unit_Vectors;
                Turn_Direction      : in  Turning_Directions;
                Turn_Status         : in  Turning_Statuses;
                Ground_Velocity     : in  Velocity_Vectors;
                Turn_Radius         : in  Segment_Distances;
                Crosstrack_Error    : out Segment_Distances;
                Heading_Error       : out Angles) is separate;

end Crosstrack_and_Heading_Error_Operations;
```

3.3.4.1.9.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table describes the data types required by this part and defined in the package specification of Waypoint_Steering:

Name	Range	Description
Turning_Directions	Left_Turn, Right_Turn	Indicates if the missile needs to make a right or a left-hand turn to go to the next waypoint
Turning_Statuses	Turning, Not_Turning	Indicates whether or not the missile is currently turning

3.3.4.1.9.2.8 LIMITATIONS

None.

3.3.4.1.9.2.9 LLCSC DESIGN

None.

3.3.4.1.9.2.10 UNIT DESIGN

3.3.4.1.9.2.10.1 COMPUTE_WHEN_TURNING (PROCEDURE BODY) UNIT DESIGN

This part computes the crosstrack and heading error for a missile in turning flight.

NOTE: By the time this part is called the waypoints have been updated so that the missile is now turning past waypoint A to go on to waypoint B.

3.3.4.1.9.2.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R173.

3.3.4.1.9.2.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.4.1.9.2.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Distance_to_B	Segment_Distances	in	Distance from missile position to the current waypoint, B
Nonturning_Distance	Segment_Distance	in	Distance from point of tangency of turn circle and next course segment, BC, to next waypoint, C
Unit_Radial_M	Unit_Vectors	in	Unit radial vector to the missile extending outward from the origin of the Earth-centered reference frame
Unit_Normal_B	Unit_Vectors	in	Segment AB unit normal vector
Unit_Tangent_B	Unit_Vectors	in	Segment AB unit tangent vector
Turn_Direction	Turning_Directions	in	Direction of turn required to go from waypoint B to waypoint C
Ground_Velocity	Velocity_Vectors	in	Missile ground velocity with N and E components
Turn_Radius	Segment_Distances	in	Desired missile turn radius
Crosstrack_Error	Segment_Distances	out	Missile displacement normal to the commanded ground track; is positive when missile is to the right as viewed in the direction of flight
Heading_Error	Angles	out	Difference between the current missile heading and the desired heading

3.3.4.1.9.2.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Crosstrack_ Distance	Segment_ Distances	Distance of missile from line segment AB
D_Y	Segment_ Distances	Difference between turning radius and crosstrack_distance
D_X	Segment_ Distances	Difference between distance to B and nonturning distance
D_H_D_C	Segment_ Distances	Used for intermediate calculations
Direction	INTEGER	+1 if missile is making a right turn, -1 if missile is making a left turn
Dot_Prod_ Result	Sin_Cos_Ratio	Dot product result, which, due to the geometry, equals the angle between the desired UR_M and the actual UR_M
N_H_D_C	Segment_ Distances	Used for intermediate calculations
R_M	Segment_ Distances	Turning radius actually being flown by the missile

3.3.4.1.9.2.10.1.5 PROCESS CONTROL

Not applicable.

3.3.4.1.9.2.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Waypoint_Steering.Crosstrack_and_Heading_Error_Operations)
procedure Compute_When_Turning
             (Distance_to_B       : in  Segment_Distances;
              Nonturning_Distance : in  Segment_Distances;
              Unit_Radial_M       : in  Unit_Vectors;
              Unit_Normal_B       : in  Unit_Vectors;
              Unit_Tangent_B      : in  Unit_Vectors;
              Turn_Direction      : in  Turning_Directions;
              Ground_Velocity     : in  Velocity_Vectors;
              Turn_Radius         : in  Segment_Distances;
              Crosstrack_Error    : out Segment_Distances;
              Heading_Error       : out Angles) is

-- ----------------------
-- --declaration section
-- ----------------------

   Crosstrack_Distance  : Segment_Distances;
   D_Y                  : Segment_Distances;
   D_X                  : Segment_Distances;
   D_H_D_C              : Segment_Distances;
   Direction            : INTEGER;
   Dot_Prod_Result      : Sin_Cos_Ratio;
   N_H_D_C              : Segment_Distances;
   R_M                  : Segment_Distances;
```

```
---------------------------------------
--begin procedure Compute_When_Turning-
---------------------------------------

begin

-- --convert turn direction to an integer value

   if Turn_Direction = Left_Turn then
      Direction := -1;
   else
      Direction := 1;
   end if;

-- --get the sine of the angle (which approximately equals the angle)
-- --between the actual and desired UR_M,
-- --and then compute crosstrack distance

   Dot_Prod_Result     := Dot_Product (Left  => Unit_Radial_M,
                                       Right => Unit_Normal_B);
   Crosstrack_Distance := Dot_Prod_Result * Earth_Radius;

-- --compute the radius of the circle that the missile is actually traversing

   D_Y := Turn_Radius - Crosstrack_Distance * Direction;
   D_X := Distance_to_B - Nonturning_Distance;
   R_M := Sqrt(D_X * D_X + D_Y * D_Y);

-- --compute crosstrack error

   Crosstrack_Error := (Turn_Radius - R_M) * Direction;

-- --compute heading error

   N_H_D_C := Direction * Unit_Tangent_B(Z) * D_X -
              Unit_Normal_B(Z) * D_Y;
   D_H_D_C := -Direction * Unit_Normal_B(Z) * D_X -
              Unit_Tangent_B(Z) * D_Y;

   Heading_Error := Arctan((N_H_D_C * Ground_Velocity(N) -
                            D_H_D_C * Ground_Velocity(E)) /
                           (D_H_D_C * Ground_Velocity(N) +
                            N_H_D_C * Ground_Velocity(E)));

end Compute_When_Turning;
```

3.3.4.1.9.2.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following subprograms are required by this part and defined as generic formal parameters to the Crosstrack_and_Heading_Error_Operations package:

Name	Type	Description
"*"	function	Multiplication operator defining the operation: Sin_Cos_Ratio * Earth_Distances => Segment_Distances
"*"	function	Multiplication operator defining the operation: Sin_Cos_Ratio * Segment_Distances => Segment_Distances
"*"	function	Multiplication operator defining the operation: Segment_Distances * INTEGER => Segment_Distances
"*"	function	Multiplication operator defining the operation: INTEGER * Sin_Cos_Ratio => Sin_Cos_Ratio
"*"	function	Multiplication operator defining the operation: Segment_Distances * Velocities => Tan_Ratio
"*"	function	Multiplication operator defining the operation: Sin_Cos_Ratio * Velocities => Velocities
"/"	function	Division operator defining the operation: Velocities / Velocities => Tan_Ratio
Dot_Product	function	Dot product function
Sqrt	function	Square root function
Arctan	function	Arctangent function

Data types:

The following data types are required by this part and defined as generic formal parameters to the Crosstrack_and_Heading_Error_Operations package:

Name	Type	Description
Navigation_Indices	discrete type	Data type used to dimension Velocity_ Vectors
Unit_Indices	discrete type	Data type used to dimension Unit_Vectors
Angles	floating point type	Data type of angular measurements
Earth_Distances	floating point type	Data type used to define distance measurements of the Earth's radius
Segment_ Distances	floating point type	Data type used to define distance measurements of the navigation segments
Sin_Cos_Ratio	floating point type	Data type of results of sine/cosine operations
Tan_Ratio	floating point type	Data type of tangent operations
Velocities	floating point type	Data type of velocity measurements
Unit_Vectors	array	Array, dimensioned by Unit_Indices, of Sin_Cos_Ratio
Velocity_Vectors	array	Array, dimensioned by Navigation_ Indices, of Velocities

The following table describes the data types required by this part and defined in the package specification of Waypoint_Steering:

Name	Range	Description
Turning_Directions	Left_Turn, Right_Turn	Indicates if the missile needs to make a right or a left-hand turn to go to the next waypoint
Turning_Statuses	Turning, Not_Turning	Indicates whether or not the missile is currently turning

Data objects:

The following data objects are required by this part and defined as generic formal parameters to the Crosstrack_and_Heading_Error_Operations package:

Name	Type	Value	Description
E	Navigation_Indices	'FIRST	Used to access first element of arrays dimensioned by Navigation_Indices
N	Navigation_Indices	'SUCC(E)	Used to access second element of arrays dimensioned by Navigation_Indices
U	Navigation_Indices	'LAST	Used to access last element of arrays dimensioned by Navigation_Indices
X	Unit_Indices Indices	'FIRST	Used to access first element of arrays dimensioned by Unit_Indices
Y	Unit_Indices Indices	'SUCC(X)	Used to access second element of arrays dimensioned by Unit_Indices
Z	Unit_Indices Indices	'LAST	Used to access last element of arrays dimensioned by Unit_Indices
Earth_Radius	Earth_Distances	n/a	Radius of the Earth

3.3.4.1.9.2.10.1.8 LIMITATIONS

None.

3.3.4.1.9.2.10.2 COMPUTE_WHEN_NOT_TURNING (PROCEDURE BODY) UNIT DESIGN

This part computes the crosstrack and heading error for a missile in nonturning flight.

3.3.4.1.9.2.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R175.

3.3.4.1.9.2.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.4.1.9.2.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Unit_Radial_M	Unit_Vectors	in	Unit radial vector to the missile extending outward from the origin of the Earth-centered reference frame
Unit_Normal_B	Unit_Vectors	in	Segment AB unit normal vector
Ground_Velocity	Velocity_Vectors	in	Missile ground velocity with N and E components
Crosstrack_Error	Segment_Distances	out	Missile displacement normal to the commanded ground track; is positive when missile is to the right as viewed in the direction of flight
Heading_Error	Angles	out	Difference between the current missile heading and the desired heading

3.3.4.1.9.2.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
D_H_D_C	Sin_Cos_Ratio	Used for intermediate calculations
Dot_Prod_Result	Sin_Cos_Ratio	Dot product result, which, due to the geometry, equals the angle between the desired UR_M and the actual UR_M
N_H_D_C	Sin_Cos_Ratio	Used for intermediate calculations

3.3.4.1.9.2.10.2.5 PROCESS CONTROL

Not applicable.

3.3.4.1.9.2.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Waypoint_Steering.Crosstrack_and_Heading_Error_Operations)
procedure Compute_When_Not_Turning
             (Unit_Radial_M    : in  Unit_Vectors;
              Unit_Normal_B    : in  Unit_Vectors;
              Ground_Velocity  : in  Velocity_Vectors;
              Crosstrack_Error : out Segment_Distances;
              Heading_Error    : out Angles) is

-- ---------------------
-- --local declarations-
```

```
-- --------------------

   Dot_Prod_Result : Sin_Cos_Ratio;
   D_H_D_C         : Sin_Cos_Ratio;
   N_H_D_C         : Sin_Cos_Ratio;

-----------------------------------------
--begin procedure Compute_When_Not_Turning-
-----------------------------------------

begin

-- --get the sine of the angle (which approximately equals the angle)
-- --between the actual and desired UR_M,
-- --and then compute crosstrack distance/error

   Dot_Prod_Result  := Dot_Product (Left  => Unit_Radial_M,
                                   Right => Unit_Normal_B);
   Crosstrack_Error := Dot_Prod_Result * Earth_Radius;

-- --compute heading error

   N_H_D_C := - Unit_Normal_B(Z);
   D_H_D_C := Unit_Normal_B(Y) * Unit_Radial_M(X) -
              Unit_Normal_B(X) * Unit_Radial_M(Y);

   Heading_Error := Arctan((N_H_D_C * Ground_Velocity(N) -
                            D_H_D_C * Ground_Velocity(E)) /
                           (D_H_D_C * Ground_Velocity(N) +
                            N_H_D_C * Ground_Velocity(E)));

end Compute_When_Not_Turning;
```

3.3.4.1.9.2.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following subprograms are required by this part and defined as generic formal parameters to the Crosstrack_and_Heading_Error_Operations package:

Name	Type	Description
"*"	function	Multiplication operator defining the operation: Sin_Cos_Ratio * Earth_Distances => Segment_Distances
"*"	function	Multiplication operator defining the operation: Sin_Cos_Ratio * Segment_Distances => Segment_Distances
"*"	function	Multiplication operator defining the operation: Sin_Cos_Ratio * Velocities => Velocities
"/"	function	Division operator defining the operation: Velocities / Velocities => Tan_Ratio
Dot_Product	function	Dot product function
Arctan	function	Arctangent function

Data types:

The following data types are required by this part and defined as generic formal parameters to the Crosstrack_and_Heading_Error_Operations package:

Name	Type	Description
Navigation_Indices	discrete type	Data type used to dimension Velocity_Vectors
Unit_Indices	discrete type	Data type used to dimension Unit_Vectors
Angles	floating point type	Data type of angular measurements
Earth_Distances	floating point type	Data type used to define distance measurements of the Earth's radius
Segment_Distances	floating point type	Data type used to define distance measurements of the navigation segments
Sin_Cos_Ratio	floating point type	Data type of results of sine/cosine operations
Tan_Ratio	floating point type	Data type of tangent operations
Velocities	floating point type	Data type of velocity measurements
Unit_Vectors	array	Array, dimensioned by Unit_Indices, of Sin_Cos_Ratio
Velocity_Vectors	array	Array, dimensioned by Navigation_Indices, of Velocities

Data objects:

The following data objects are required by this part and defined as generic formal parameters to the Crosstrack_and_Heading_Error_Operations package:

Name	Type	Value	Description
E	Navigation_Indices	'FIRST	Used to access first element of arrays dimensioned by Navigation_Indices
N	Navigation_Indices	'SUCC(E)	Used to access second element of arrays dimensioned by Navigation_Indices
U	Navigation_Indices	'LAST	Used to access last element of arrays dimensioned by Navigation_Indices
X	Unit_Indices Indices	'FIRST	Used to access first element of arrays dimensioned by Unit_Indices
Y	Unit_Indices Indices	'SUCC(X)	Used to access second element of arrays dimensioned by Unit_Indices
Z	Unit_Indices Indices	'LAST	Used to access last element of arrays dimensioned by Unit_Indices
Earth_Radius	Earth_Distances	n/a	Radius of the Earth

3.3.4.1.9.2.10.2.8 LIMITATIONS

None.

3.3.4.1.9.2.10.3 COMPUTE (PROCEDURE BODY) UNIT DESIGN

This part computes the crosstrack and heading error for a missile in turning or nonturning flight.

3.3.4.1.9.2.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R174.

3.3.4.1.9.2.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.4.1.9.2.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Distance_to_B	Segment_Distances	in	Distance from missile position to the current waypoint, B
Nonturning_Distance	Segment_Distances	in	Distance from point of tangency of turn circle and next course segment, BC, to next waypoint, C
Unit_Radial_M	Unit_Vectors	in	Unit radial vector to the missile extending outward from the origin of the Earth-centered reference frame
Unit_Normal_B	Unit_Vectors	in	Segment AB unit normal vector
Unit_Tangent_B	Unit_Vectors	in	Segment AB unit tangent vector
Turn_Direction	Turning_Directions	in	Direction of turn required to go from waypoint B to waypoint C
Turn_Status	Turning_Statuses	in	Indicates if the missile is in turning or nonturning flight
Ground_Velocity	Velocity_Vectors	in	Missile ground velocity with N and E components
Turn_Radius	Segment_Distances	in	Desired missile turn radius
Crosstrack_Error	Segment_Distances	out	Missile displacement normal to the commanded ground track; is positive when missile is to the right as viewed in the direction of flight
Heading_Error	Angles	out	Difference between the current missile heading and the desired heading

3.3.4.1.9.2.10.3.4 LOCAL DATA

None.

3.3.4.1.9.2.10.3.5 PROCESS CONTROL

Not applicable.

3.3.4.1.9.2.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Waypoint_Steering.Crosstrack_and_Heading_Error_Operations)
procedure Compute
            (Distance_to_B       : in  Segment_Distances;
             Nonturning_Distance : in  Segment_Distances;
             Unit_Radial_M       : in  Unit_Vectors;
             Unit_Normal_B       : in  Unit_Vectors;
             Unit_Tangent_B      : in  Unit_Vectors;
             Turn_Direction      : in  Turning_Directions;
             Turn_Status         : in  Turning_Statuses;
```

```
                Ground_Velocity     : in  Velocity_Vectors;
                Turn_Radius         : in  Segment_Distances;
                Crosstrack_Error    : out Segment_Distances;
                Heading_Error       : out Angles) is
begin
   if Turn_Status = Turning then
      Compute_When_Turning
                   (Distance_to_B       => Distance_to_B       ,
                    Nonturning_Distance => Nonturning_Distance ,
                    Unit_Radial_M       => Unit_Radial_M       ,
                    Unit_Normal_B       => Unit_Normal_B       ,
                    Unit_Tangent_B      => Unit_Tangent_B      ,
                    Turn_Direction      => Turn_Direction      ,
                    Ground_Velocity     => Ground_Velocity     ,
                    Turn_Radius         => Turn_Radius         ,
                    Crosstrack_Error    => Crosstrack_Error    ,
                    Heading_Error       => Heading_Error       );
   else
      Compute_When_Not_Turning
                   (Unit_Radial_M    => Unit_Radial_M    ,
                    Unit_Normal_B    => Unit_Normal_B    ,
                    Ground_Velocity  => Ground_Velocity  ,
                    Crosstrack_Error => Crosstrack_Error ,
                    Heading_Error    => Heading_Error    );
   end if;
end Compute;
```

3.3.4.1.9.2.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and contained in the Crosstrack_and_Heading_Error_Operations package.

Name	Type	Description
Compute_When_Turning	procedure	Computes the crosstrack and heading errors when the missile is in turning flight
Compute_When_Not_Turning	procedure	Computes the crosstrack and heading errors when the missile is in nonturning flight

Data types:

The following data types are required by this part and defined as generic formal parameters to the Crosstrack_and_Heading_Error_Operations package:

Name	Type	Description
Navigation_Indices	discrete type	Data type used to dimension Velocity_ Vectors
Unit_Indices	discrete type	Data type used to dimension Unit_Vectors
Angles	floating point type	Data type of angular measurements
Earth_Distances	floating point type	Data type used to define distance measurements of the Earth's radius
Segment_ Distances	floating point type	Data type used to define distance measurements of the navigation segments
Sin_Cos_Ratio	floating point type	Data type of results of sine/cosine operations
Tan_Ratio	floating point type	Data type of tangent operations
Velocities	floating point type	Data type of velocity measurements
Unit_Vectors	array	Array, dimensioned by Unit_Indices, of Sin_Cos_Ratio
Velocity_Vectors	array	Array, dimensioned by Navigation_ Indices, of Velocities

The following table describes the data types required by this part and defined in the package specification of Waypoint_Steering:

Name	Range	Description
Turning_Directions	Left_Turn, Right_Turn	Indicates if the missile needs to make a right or a left-hand turn to go to the next waypoint
Turning_Statuses	Turning, Not_Turning	Indicates whether or not the missile is currently turning

3.3.4.1.9.2.10.3.8 LIMITATIONS

None.

3.3.4.1.9.3 TURN_TEST_OPERATIONS (PACKAGE BODY) PACKAGE DESIGN (CATALOG #P112-0)

This part contains the operations required to determine if a missile should be in turning or nonturning flight.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.4.1.9.3.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
Stop_Test	R178
Start_Test	R179

3.3.4.1.9.3.2 LOCAL ENTITIES DESIGN

None.

3.3.4.1.9.3.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined when this part was specified:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Distances	floating	Data type of distance measurements

3.3.4.1.9.3.4 LOCAL DATA

None.

3.3.4.1.9.3.5 PROCESS CONTROL

Not applicable.

3.3.4.1.9.3.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Waypoint_Steering)
package body Turn_Test_Operations is

   function Stop_Test
               (Distance_to_B       : Distances;
                Nonturning_Distance : Distances;
                Lead_Distance       : Distances)
```

```
            return Turning_Statuses is separate;

   function Start_Test
            (Distance_to_B       : Distances;
             Turning_Distance    : Distances;
             Lead_Distance       : Distances)
            return Turning_Statuses is separate;

end Turn_Test_Operations;
```

3.3.4.1.9.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table summarizes the types required by this part and defined elsewhere in the parent top level component:

Name	Range	Description
Turning_Statuses	Turning, Not_Turning	Indicates whether or not the missile is currently turning

3.3.4.1.9.3.8 LIMITATIONS

None.

3.3.4.1.9.3.9 LLCSC DESIGN

None.

3.3.4.1.9.3.10 UNIT DESIGN

3.3.4.1.9.3.10.1 STOP_TEST (FUNCTION BODY) UNIT DESIGN

This part determines whether a missile should be in turning or nonturning flight.

3.3.4.1.9.3.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R178.

3.3.4.1.9.3.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.4.1.9.3.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Distance_to_B	Distances	In	Distance from missile position to current waypoint, B
Nonturning_Distance	Distances	In	Distance from point of tangency of turn circle and next course segment, BC, to the next waypoint, C
Lead_Distance	Distances	In	Distance at which a turn is started or stopped early to compensate for the delay in missile roll dynamics

3.3.4.1.9.3.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Turn_Status	Turning_Statuses	N/A	Turning or nonturning status of the missile

3.3.4.1.9.3.10.1.5 PROCESS CONTROL

Not applicable.

3.3.4.1.9.3.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Waypoint_Steering.Turn_Test_Operations)
function Stop_Test
           (Distance_to_B       : Distances;
            Nonturning_Distance : Distances;
            Lead_Distance       : Distances) return Turning_Statuses is

-- ---------------------
-- --declaration section-
-- ---------------------
```

```
   Turn_Status : Turning_Statuses := Turning;

---------------------------
--begin function Stop_Test-
---------------------------

begin

   if Distance_to_B <= (Lead_Distance + Nonturning_Distance) then
      Turn_Status := Not_Turning;
   end if;

   return Turn_Status;

end Stop_Test;
```

3.3.4.1.9.3.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table summarizes the types required by this part and defined as generic parameters to the Turn_Test_Operations package:

Name	Type	Description
Distances	floating	Data type of distance measurements

The following table summarizes the types required by this part and defined in the package specification of Waypoint_Steering:

Name	Range	Description
Turning_Statuses	Turning, Not_Turning	Indicates whether or not the missile is currently turning

3.3.4.1.9.3.10.1.8 LIMITATIONS

None.

3.3.4.1.9.3.10.2 START_TEST (FUNCTION BODY) UNIT DESIGN

This part determines whether a missile should be in turning or nonturning flight.

3.3.4.1.9.3.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R179.

3.3.4.1.9.3.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.4.1.9.3.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Distance_to_B	Distances	In	Distance from missile position to current waypoint, B
Turning_Distance	Distances	In	Distance from point of tangency of turn circle and current course segment, BC, to the current waypoint, C
Lead_Distance	Distances	In	Distance at which a turn is started or stopped early to compensate for the delay in missile roll dynamics

3.3.4.1.9.3.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Turn_Status	Turning_Statuses	N/A	Turning or nonturning status of the missile

3.3.4.1.9.3.10.2.5 PROCESS CONTROL

Not applicable.

3.3.4.1.9.3.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Waypoint_Steering.Turn_Test_Operations)
function Start_Test
            (Distance_to_B       : Distances;
             Turning_Distance    : Distances;
```

```
            Lead_Distance       : Distances) return Turning_Statuses is

-- ---------------------
-- --declaration section-
-- ---------------------

   Turn_Status : Turning_Statuses := Not_Turning;

----------------------------
--begin function Start_Test-
----------------------------

begin

   if Distance_to_B <= (Lead_Distance + Turning_Distance) then
      Turn_Status := Turning;
   end if;

   return Turn_Status;

end Start_Test;
```

3.3.4.1.9.3.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following table summarizes the types required by this part and defined as generic parameters to the Turn_Test_Operations package:

Name	Type	Description
Distances	floating	Data type of distance measurements

The following table summarizes the types required by this part and defined in the package specification of Waypoint_Steering:

Name	Range	Description
Turning_Statuses	Turning, Not_Turning	Indicates whether or not the missile is currently turning

3.3.4.1.9.3.10.2.8 LIMITATIONS

None.

3.3.4.1.9.4 STEERING_VECTOR_OPERATIONS_WITH_ARCSIN (PACKAGE BODY) PACKAGE DESIGN (CATALOG #P1048-0)

This package contains operations to do the following: o Initialize the waypoint steering vectors when supplied with the latitude and longitude of the past, current, and next waypoints o Update the waypoint steering vectors when supplied with the latitude and longitude of the "new" waypoint, C.

The waypoint steering vectors for a course segment, extending from waypoint A to waypoint B, are the segment unit normal vector (UN_B) and the segment unit tangent vector (UT_B).

It does not make the assumption that alphasin(alpha) when doing its computations.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.4.1.9.4.1 REQUIREMENTS ALLOCATION

The following table summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
Initialize	N/A
Update	N/A

3.3.4.1.9.4.2 LOCAL ENTITIES DESIGN

None.

3.3.4.1.9.4.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined with the specification for this part.

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Indices	discrete type	Used to dimension Unit_Vectors
Earth_Distances	floating point type	Data type used to define distance measurements of the Earth's radius
Segment_Distances	floating point type	Data type used to define distance measurements of the navigation segments
Earth_Positions	floating point type	Data type used to define latitude and longitude measurements
Radians	floating point type	Radian units of angular measurement
Sin_Cos_Ratio	floating point type	Data type used to define results of a sine or cosine function
Unit_Vectors	array	Array of "Sin_Cos_Ratio" dimensioned by Indices

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Earth_Radius	Earth_Distances	n/a	Radius of the Earth

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplication operator defining the operation: Sin_Cos_Ratio * Earth_Distances => Segment_Distances
"/"	function	Operator defining the operation: Unit_Vectors / Sin_Cos_Ratio => Unit_Vectors
Arcsin	function	Arcsin function (must return radians)
Cross_Product	procedure	Cross product function
Vector_Length	function	Calculates the length of a vector
Sin_Cos	procedure	Calculates the sine and cosine of an input value

3.3.4.1.9.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Unit_Radial_B	Unit_Vectors	N/A	Unit radial vector to waypoint B extending outwards from the origin of the Earth-centered reference frame towards waypoint B
Unit_Radial_C	Unit_Vectors	N/A	Unit radial vector to waypoint C extending outwards from the origin of the Earth-centered reference frame towards waypoint C

3.3.4.1.9.4.5 PROCESS CONTROL

Not applicable.

3.3.4.1.9.4.6 PROCESSING

The following describes the processing performed by this part:

```
with Geometric_Operations;
separate (Waypoint_Steering)
package body Steering_Vector_Operations_with_Arcsin is

   package Geo renames Geometric_Operations;

-- ----------------------------
-- --instantiate required parts-
-- ----------------------------

   function U_R_Vector is new
             Geo.Unit_Radial_Vector (Indices          => Indices,
                                     Earth_Positions  => Earth_Positions,
                                     Sin_Cos_Ratio    => Sin_Cos_Ratio,
                                     Unit_Vectors     => Unit_Vectors);

   procedure Compute_Segment_and_U_Nl_Vector is new
               Geo.Compute_Segment_and_Unit_Normal_Vector_with_Arcsin
                  (Indices             => Indices,
                   Earth_Distances     => Earth_Distances,
                   Segment_Distances   => Segment_Distances,
                   Radians             => Radians,
                   Sin_Cos_Ratio       => Sin_Cos_Ratio,
                   Unit_Vectors        => Unit_Vectors,
                   Earth_Radius        => Earth_Radius);

-- --------------------
-- --local declarations-
-- --------------------
```

```
   Unit_Radial_B : Unit_Vectors;
   Unit_Radial_C : Unit_Vectors;

end Steering_Vector_Operations_with_Arcsin;
```

3.3.4.1.9.4.7 UTILIZATION OF OTHER ELEMENTS

The following library units are with'd by this part:
1. Geometric_Operations package (Geo)

UTILIZATION OF EXTERNAL ELEMENTS:

Subprograms and task entries:

The following table summarizes the external subroutines and task entries required by this part:

Name	Type	Source	Description
Unit_Radial_Vector	generic function	Geo	Computes a unit radial vector
Compute_Segment_and_Unit_Normal_Vector_with_Arcsin	generic function	Geo	Computes segment distance and unit normal vector

3.3.4.1.9.4.8 LIMITATIONS

None.

3.3.4.1.9.4.9 LLCSC DESIGN

None.

3.3.4.1.9.4.10 UNIT DESIGN

3.3.4.1.9.4.10.1 INITIALIZE (PROCEDURE BODY) UNIT DESIGN

This part initializes the waypoint steering vectors when supplied with the latitude and longitude of the past (A), present (B), and next (C) waypoints.

This part initializes the waypoint steering vectors for the "current" course segment AB, as well as for the "next" course segment BC.

3.3.4.1.9.4.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R170.

3.3.4.1.9.4.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.4.1.9.4.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Waypoint_A_Lat, Waypoint_A_Long	Earth_Positions	in	Latitude and longitude of the "previous" waypoint
Waypoint_B_Lat, Waypoint_B_Long	Earth_Positions	in	Latitude and longitude of the "current" waypoint
Waypoint_C_Lat, Waypoint_C_Long	Earth_Positions	in	Latitude and longitude of the "next" waypoint
Unit_Normal_B, Unit_Normal_C	Unit_Vectors	out	Unit normal vectors for segments AB and BC
Unit_Tangent_B, Unit_Tangent_C	Unit_Vectors	out	Unit tangent vectors for segments AB and BC
Segment_BC_ Distance	Segment_ Distances	out	Great circle arclength between waypoints B and C

3.3.4.1.9.4.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Temp_UN_B	Unit_Vectors	N/A	Temporary Unit_Normal_B vector
Temp_UN_C	Unit_Vectors	N/A	Temporary Unit_Normal_C vector
Unit_Radial_A	Unit_Vectors	N/A	Unit radial vector pointing to waypoint A
V_Length	Sin_Cos_Ratio	N/A	Vector length

3.3.4.1.9.4.10.1.5 PROCESS CONTROL

Not applicable.

3.3.4.1.9.4.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   procedure Initialize
                (Waypoint_A_Lat      : in  Earth_Positions;
```

```
                Waypoint_A_Long     : in  Earth_Positions;
                Waypoint_B_Lat      : in  Earth_Positions;
                Waypoint_B_Long     : in  Earth_Positions;
                Waypoint_C_Lat      : in  Earth_Positions;
                Waypoint_C_Long     : in  Earth_Positions;
                Unit_Normal_B       : out Unit_Vectors;
                Unit_Normal_C       : out Unit_Vectors;
                Unit_Tangent_B      : out Unit_Vectors;
                Unit_Tangent_C      : out Unit_Vectors;
                Segment_BC_Distance : out Segment_Distances) is

--    ----------------------
--    --declaration section-
--    ----------------------

      Temp_UN_B     : Unit_Vectors;
      Temp_UN_C     : Unit_Vectors;
      Unit_Radial_A : Unit_Vectors;
      V_Length      : Sin_Cos_Ratio;

--   ----------------------------
--   --begin procedure Initialize-
--   ----------------------------

   begin

--    --compute unit radial vectors

      Unit_Radial_A := U_R_Vector(Lat_of_Point  => Waypoint_A_Lat,
                                  Long_of_Point => Waypoint_A_Long);
      Unit_Radial_B := U_R_Vector(Lat_of_Point  => Waypoint_B_Lat,
                                  Long_of_Point => Waypoint_B_Long);
      Unit_Radial_C := U_R_Vector(Lat_of_Point  => Waypoint_C_Lat,
                                  Long_of_Point => Waypoint_C_Long);

--    --compute UN_B

      Temp_UN_B      := Cross_Product(Left  => Unit_Radial_B,
                                      Right => Unit_Radial_A);
      V_Length       := Vector_Length(Temp_UN_B);
      Temp_UN_B      := Temp_UN_B / V_Length;
      Unit_Normal_B  := Temp_UN_B;

--    --compute UT_B

      Unit_Tangent_B := Cross_Product(Left  => Temp_UN_B,
                                      Right => Unit_Radial_B);

--    --compute UN_C and segment BC distance

      Compute_Segment_and_U_Nl_Vector
         (Unit_Radial1     => Unit_Radial_B,
          Unit_Radial2     => Unit_Radial_C,
          Unit_Normal2     => Temp_UN_C,
          Segment_Distance => Segment_BC_Distance);

      Unit_Normal_C  := Temp_UN_C;
```

```
--     --compute UT_C

       Unit_Tangent_C := Cross_Product(Left  => Temp_UN_C,
                                       Right => Unit_Radial_C);

   end Initialize;
```

3.3.4.1.9.4.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and instantiated in the body of the Steering_Vector_Operations package:

Name	Type	Description
U_R_Vector	function	Computes a unit radial vector
Compute_Segment_and_U_Nl_Vector	function	Computes segment distance and unit normal vector

The following table describes the subroutines required by this part and defined as generic formal subroutines to the Steering_Vector_Operations package:

Name	Type	Description
"/"	function	Operator defining the operation: Unit_Vectors / Sin_Cos_Ratio => Unit_Vectors
Cross_Product	procedure	Cross product function
Vector_Length	function	Calculates the length of a vector

Data types:

The following data types are required by this part and defined as generic parameters to the Steering_Vector_Operations package:

Name	Type	Description
Indices	discrete type	Used to dimension Unit_Vectors
Earth_Distances	floating point type	Data type used to define distance measurements of the Earth's radius
Segment_ Distances	floating point type	Data type used to define distance measurements of the navigation segments
Earth_Positions	floating point type	Data type used to define latitude and longitude measurements
Sin_Cos_Ratio	floating point type	Data type used to define results of a sine or cosine function
Unit_Vectors	array	Array of "Sin_Cos_Ratio" dimensioned by Indices

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Steering_Vector_Operations:

Name	Type	Value	Description
Unit_Radial_B	Unit_Vectors	N/A	Unit radial vector to waypoint B extending outwards from the origin of the Earth-centered reference frame towards waypoint B
Unit_Radial_C	Unit_Vectors	N/A	Unit radial vector to waypoint C extending outwards from the origin of the Earth-centered reference frame towards waypoint C

3.3.4.1.9.4.10.1.8 LIMITATIONS

None.

3.3.4.1.9.4.10.2 UPDATE (PROCEDURE BODY) UNIT DESIGN

This part updates the waypoint steering vectors when supplied with the latitude and longitude of the "new" waypoint, C.

The waypoint steering vectors for a course segment, extending from waypoint A to waypoint B, are the segment unit normal vector (UN_B) and the segment unit tangent vector (UT_B).

3.3.4.1.9.4.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R171.

3.3.4.1.9.4.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.4.1.9.4.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Waypoint_C_Lat, Waypoint_C_Long	Earth_Positions	in	Latitude and longitude of the "next" waypoint
Unit_Normal_B, Unit_Normal_C	Unit_Vectors	out in out	Unit normal vectors for segments AB and BC
Unit_Tangent_B, Unit_Tangent_C	Unit_Vectors	out in out	Unit tangent vectors for segments AB and BC
Segment_BC_ Distance	Segment_ Distances	out	Great circle arclength between waypoints B and C

3.3.4.1.9.4.10.2.4 LOCAL DATA

None.

3.3.4.1.9.4.10.2.5 PROCESS CONTROL

Not applicable.

3.3.4.1.9.4.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
   procedure Update
                (Waypoint_C_Lat      : in     Earth_Positions;
                 Waypoint_C_Long     : in     Earth_Positions;
                 Unit_Normal_B       :    out Unit_Vectors;
                 Unit_Normal_C       : in out Unit_Vectors;
                 Unit_Tangent_B      :    out Unit_Vectors;
                 Unit_Tangent_C      : in out Unit_Vectors;
                 Segment_BC_Distance :    out Segment_Distances) is
   begin

--     --advance "C" vectors into "B" vectors

       Unit_Radial_B  := Unit_Radial_C;
```

```
      Unit_Normal_B  := Unit_Normal_C;
      Unit_Tangent_B := Unit_Tangent_C;

--    --calculate new values

      Unit_Radial_C  := U_R_Vector(Lat_of_Point  => Waypoint_C_Lat,
                                   Long_of_Point => Waypoint_C_Long);

      Compute_Segment_and_U_Nl_Vector
         (Unit_Radial1     => Unit_Radial_B,
          Unit_Radial2     => Unit_Radial_C,
          Unit_Normal2     => Unit_Normal_C,
          Segment_Distance => Segment_BC_Distance);

      Unit_Tangent_C := Cross_Product(Left  => Unit_Normal_C,
                                      Right => Unit_Radial_C);

   end Update;
```

3.3.4.1.9.4.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and instantiated in the body of the Steering_Vector_Operations package:

Name	Type	Description
U_R_Vector	function	Computes a unit radial vector
Compute_Segment_and_U_Nl_Vector	function	Computes segment distance and unit normal vector

The following table describes the subroutines required by this part and defined as generic formal subroutines to the Steering_Vector_Operations package:

Name	Type	Description
Cross_Product	procedure	Cross product function

Data types:

The following data types are required by this part and defined as generic parameters to the Steering_Vector_Operations package:

Name	Type	Description
Indices	discrete type	Used to dimension Unit_Vectors
Earth_Distances	floating point type	Data type used to define distance measurements of the Earth's radius
Segment_ Distances	floating point type	Data type used to define distance measurements of the navigation segments
Earth_Positions	floating point type	Data type used to define latitude and longitude measurements
Sin_Cos_Ratio	floating point type	Data type used to define results of a sine or cosine function
Unit_Vectors	array	Array of "Sin_Cos_Ratio" dimensioned by Indices

Data objects:

The following table summarizes the objects required by this part and defined in the package body of Steering_Vector_Operations:

Name	Type	Value	Description
Unit_Radial_B	Unit_Vectors	N/A	Unit radial vector to waypoint B extending outwards from the origin of the Earth-centered reference frame towards waypoint B
Unit_Radial_C	Unit_Vectors	N/A	Unit radial vector to waypoint C extending outwards from the origin of the Earth-centered reference frame towards waypoint C

3.3.4.1.9.4.10.2.8 LIMITATIONS

None.

3.3.4.1.10 UNIT DESIGN

3.3.4.1.10.1 COMPUTE_TURN_ANGLE_AND_DIRECTION (PROCEDURE BODY) UNIT DESIGN (CATALOG #P108-0)

Using the waypoint steering vectors, this part computes the tangent of one-half the turn angle along with the turn direction.

3.3.4.1.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R172.

3.3.4.1.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.4.1.10.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were defined when this part was previously specified.

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Unit_Vectors	private	One-dimensional, three-element arrays defining the waypoint steering vectors
Sin_Cos_Ratio	floating point type	Data type of results of sine/cosine operations
Tan_Ratio	floating point type	Data type of results of tangent operations

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"+"	function	Addition operator defining the operation: Tan_Ratio + Sin_Cos_Ratio => Tan_Ratio
"/"	function	Division operator defining the operation: Sin_Cos_Ratio / Tan_Ratio => Tan_Ratio
Dot_Product	function	Calculates the dot product of two Unit_Vectors

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Unit_Normal_C	Unit_Vectors	in	Segment BC unit normal vector with x, y, and z components
Unit_Tangent_B	Unit_Vectors	in	Segment AB unit tangent vector with x, y, and z components
Unit_Tangent_C	Unit_Vectors	in	Segment BC unit tangent vector with x, y, and z components
Tan_of_One_Half_Turn_Angle	Tan_Ratio	out	Tangent of one-half the angle between the current course segment and the next course segment
Turn_Direction	Turning_Directions	out	Indicates if missile is to make a right- or left-hand turn

3.3.4.1.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Cos_of_Turn_Angle, Sin_of_Turn_Angle	Sin_Cos_Ratio	N/A	Sine and cosine of the angle between the current and next course segments

3.3.4.1.10.1.5 PROCESS CONTROL

Not applicable.

3.3.4.1.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Waypoint_Steering)
procedure Compute_Turn_Angle_and_Direction
             (Unit_Normal_C              : in     Unit_Vectors;
              Unit_Tangent_B             : in     Unit_Vectors;
              Unit_Tangent_C             : in     Unit_Vectors;
              Tan_of_One_Half_Turn_Angle :    out Tan_Ratio;
              Turn_Direction             :    out Turning_Directions) is

-- ----------------------
-- --declaration section-
-- ----------------------

   Tan_of_Half       : Tan_Ratio renames Tan_of_One_Half_Turn_Angle;
```

```
   Cos_of_Turn_Angle : Sin_Cos_Ratio;
   Sin_of_Turn_Angle : Sin_Cos_Ratio;

-------------------------------------------------
--begin procedure Compute_Turn_Angle_and_Direction-
-------------------------------------------------

begin

   Cos_of_Turn_Angle := Dot_Product(Left  => Unit_Tangent_B,
                                    Right => Unit_Tangent_C);
   Sin_of_Turn_Angle := Dot_Product(Left  => Unit_Tangent_B,
                                    Right => Unit_Normal_C);

   Tan_of_Half := ABS(Sin_of_Turn_Angle /
                      (Tan_Ratio(1.0) + Cos_of_Turn_Angle));

   if Sin_of_Turn_Angle < 0.0 then
      Turn_Direction := Left_Turn;
   else
      Turn_Direction := Right_Turn;
   end if;

end Compute_Turn_Angle_and_Direction;
```

3.3.4.1.10.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.4.1.10.1.8 LIMITATIONS

None.

3.3.4.1.10.2 DISTANCE_TO_CURRENT_WAYPOINT (FUNCTION BODY) UNIT DESIGN (CATALOG #P110-0)

This part computes the distance from the missile's position to the current waypoint, B.

3.3.4.1.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R176.

3.3.4.1.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.4.1.10.2.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined with this part's specification.

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Unit_Vectors	private	One dimensional, three-element array of Sin_Cos_Ratio
Sin_Cos_Ratio	floating point type	Results of sine/cosine operations
Tan_Ratio	floating point type	Results of tangent operation
Earth_Distances	floating point type	Data type used to define distance measurements of the Earth's radius
Segment_Distances	floating point type	Data type used to define distance measurements of the navigation segments

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Earth_Radius	Earth_Distances	n/a	Radius of the Earth

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
Dot_Product	function	Computes the dot product of two unit vectors
"*"	function	Multiplication operator defining the operation: Sin_Cos_Ratio * Earth_Distances => Segment_Distances

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Unit_Radial_M	Unit_Vectors	in	Unit radial vector to the missile extending outward from the origin of the Earth-centered reference frame
Unit_Tangent_B	Unit_Vectors	in	Segment AB unit tangent vector

3.3.4.1.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Dot_Prod_Result	Sin_Cos_Ratio	N/A	Due to the geometry, this value ends up being the angle between UR_M and UR_B

3.3.4.1.10.2.5 PROCESS CONTROL

Not applicable.

3.3.4.1.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Waypoint_Steering)
function Distance_to_Current_Waypoint
            (Unit_Radial_M  : Unit_Vectors;
             Unit_Tangent_B : Unit_Vectors) return Segment_Distances is

-- ---------------------
-- --local declarations-
-- ---------------------

   Dot_Prod_Result : Sin_Cos_Ratio;

-----------------------------------------------
--begin function Distance_To_Current_Waypoint-
-----------------------------------------------

begin

-- --dot product of UR_M and UT_B, because of the geometry, equals
-- --the angle between UR_M and UR_B
   Dot_Prod_Result := Dot_Product (Left  => Unit_Radial_M,
                                   Right => Unit_Tangent_B);
```

```
   return Dot_Prod_Result * Earth_Radius;
end Distance_to_Current_Waypoint;
```

3.3.4.1.10.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.4.1.10.2.8 LIMITATIONS

None.

3.3.4.1.10.3 COMPUTE_TURNING_AND_NONTURNING_DISTANCES(PROCEDURE BODY) UNIT DESIGN (CATALOG #P111-0)

This part computes the missile turning distance projected onto the current course segment, AB, and the missile nonturning distance measured along the next course segment, BC.

3.3.4.1.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R177.

3.3.4.1.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.4.1.10.3.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined when this part was specified.

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Distances	floating point type	Data type of distance measurements
Tan_Ratio	floating point type	Data type of results of tangent operation

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplication operator defining the operation: Distances * Tan_Ratio => Distances

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Tan_of_One_Half_Turn_Angle	Tan_Ratio	in	Tangent of 1/2 the angle between current course segment and next course segment
Segment_BC_Distance	Distances	in	Great circle arc length between waypoints B and C
Turn_Radius	Distances	in	Desired missile turn radius
Turning_Distance	Distances	out	Distance from the point of tangency of the turn circle and the current course segment AB to the current waypoint, B
Nonturning_Distance	Distances	out	Distance from the point of tangency of the turn circle and the next course segment BC to the next waypoint, C

3.3.4.1.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Temp_Turning_Distance	Distances	N/A	Temporary variable

3.3.4.1.10.3.5 PROCESS CONTROL

Not applicable.

3.3.4.1.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Waypoint_Steering)
procedure Compute_Turning_and_Nonturning_Distances
             (Tan_of_One_Half_Turn_Angle : in     Tan_Ratio;
              Segment_BC_Distance        : in     Distances;
              Turn_Radius                : in     Distances;
```

```
                Turning_Distance          :    out Distances;
                Nonturning_Distance       :    out Distances) is

-- ---------------------
-- --declaration section-
-- ---------------------

   Temp_Turning_Distance : Distances;

------------------------------------------------------------
--begin procedure Compute_Turning_And_Nonturning_Distances-
------------------------------------------------------------

begin

   Temp_Turning_Distance := Turn_Radius * Tan_of_One_Half_Turn_Angle;
   Turning_Distance      := Temp_Turning_Distance;
   Nonturning_Distance   := Segment_BC_Distance -  Temp_Turning_Distance;

end Compute_Turning_And_Nonturning_Distances;
```

3.3.4.1.10.3.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.4.1.10.3.8 LIMITATIONS

None.

3.3.4.1.10.4 DISTANCE_TO_CURRENT_WAYPOINT_WITH_ARCSIN (FUNCTION BODY) UNIT DESIGN (CATALOG #P1117-0)

This part computes the distance from the missile's position to the current waypoint, B.

It does not use the assumption that alphasin(alpha) when doing its computations.

3.3.4.1.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R.

3.3.4.1.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.4.1.10.4.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously defined with this part's specification.

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Unit_Vectors	private	One dimensional, three-element array of Sin_Cos_Ratio
Radians	floating point type	Radian units of angular measurement
Sin_Cos_Ratio	floating point type	Results of sine/cosine operations
Tan_Ratio	floating point type	Results of tangent operation
Earth_Distances	floating point type	Data type used to define distance measurements of the Earth's radius
Segment_ Distances	floating point type	Data type used to define distance measurements of the navigation segments

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Earth_Radius	Earth_ Distances	n/a	Radius of the Earth

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
Dot_Product	function	Computes the dot product of two unit vectors
"*"	function	Multiplication operator defining the operation: Radians * Earth_Distances => Segment_Distances

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Unit_Radial_M	Unit_Vectors	in	Unit radial vector to the missile extending outward from the origin of the Earth-centered reference frame
Unit_Tangent_B	Unit_Vectors	in	Segment AB unit tangent vector

3.3.4.1.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Dot_Prod_Result	Sin_Cos_Ratio	N/A	Due to the geometry, this value ends up being the angle between UR_M and UR_B

3.3.4.1.10.4.5 PROCESS CONTROL

Not applicable.

3.3.4.1.10.4.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Waypoint_Steering)
function Distance_to_Current_Waypoint_with_Arcsin
            (Unit_Radial_M  : Unit_Vectors;
             Unit_Tangent_B : Unit_Vectors) return Segment_Distances is

-- ---------------------
-- --local declarations-
-- ---------------------

   Dot_Prod_Result : Sin_Cos_Ratio;

-----------------------------------------------
--begin function Distance_To_Current_Waypoint-
-----------------------------------------------

begin

-- --dot product of UR_M and UT_B, because of the geometry, equals
-- --the angle between UR_M and UR_B
   Dot_Prod_Result := Dot_Product (Left  => Unit_Radial_M,
                                   Right => Unit_Tangent_B);
```

```
   return Arcsin(Dot_Prod_Result) * Earth_Radius;
end Distance_to_Current_Waypoint_with_Arcsin;
```

3.3.4.1.10.4.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.4.1.10.4.8 LIMITATIONS

None.

(This page left intentionally blank.)

```
package body Waypoint_Steering is

   package body Steering_Vector_Operations is separate;

   package body Steering_Vector_Operations_With_Arcsin is separate;

   procedure Compute_Turn_Angle_And_Direction
               (Unit_Normal_C              : in     Unit_Vectors;
                Unit_Tangent_B             : in     Unit_Vectors;
                Unit_Tangent_C             : in     Unit_Vectors;
                Tan_Of_One_Half_Turn_Angle :    out Tan_Ratio;
                Turn_Direction             :    out Turning_Directions)
               is separate;

   package body Crosstrack_And_Heading_Error_Operations is separate;

   function Distance_To_Current_Waypoint
              (Unit_Radial_M  : Unit_Vectors;
               Unit_Tangent_B : Unit_Vectors)
              return Segment_Distances is separate;

   function Distance_To_Current_Waypoint_With_Arcsin
              (Unit_Radial_M  : Unit_Vectors;
               Unit_Tangent_B : Unit_Vectors)
              return Segment_Distances is separate;

   procedure Compute_Turning_And_Nonturning_Distances
               (Tan_Of_One_Half_Turn_Angle : in     Tan_Ratio;
                Segment_Bc_Distance        : in     Distances;
                Turn_Radius                : in     Distances;
                Turning_Distance           :    out Distances;
                Nonturning_Distance        :    out Distances) is separate;

   package body Turn_Test_Operations is separate;

end Waypoint_Steering;
```

```
with Geometric_Operations;
separate (Waypoint_Steering)
package body Steering_Vector_Operations is

   package Geo renames Geometric_Operations;

-- ------------------------------
-- --instantiate required parts-
-- ------------------------------

   function U_R_Vector is new
               Geo.Unit_Radial_Vector (Indices          => Indices,
                                       Earth_Positions => Earth_Positions,
                                       Sin_Cos_Ratio   => Sin_Cos_Ratio,
                                       Unit_Vectors    => Unit_Vectors);

   procedure Compute_Segment_And_U_Nl_Vector is new
                Geo.Compute_Segment_And_Unit_Normal_Vector
                   (Indices           => Indices,
                    Earth_Distances    => Earth_Distances,
                    Segment_Distances  => Segment_Distances,
                    Sin_Cos_Ratio      => Sin_Cos_Ratio,
                    Unit_Vectors       => Unit_Vectors,
                    Earth_Radius       => Earth_Radius);

-- ---------------------
-- --local declarations-
-- ---------------------

   Unit_Radial_B : Unit_Vectors;
   Unit_Radial_C : Unit_Vectors;

-- ----------------------
-- --separate procedures-
-- ----------------------

   procedure Initialize
               (Waypoint_A_Lat      : in  Earth_Positions;
                Waypoint_A_Long     : in  Earth_Positions;
                Waypoint_B_Lat      : in  Earth_Positions;
                Waypoint_B_Long     : in  Earth_Positions;
                Waypoint_C_Lat      : in  Earth_Positions;
                Waypoint_C_Long     : in  Earth_Positions;
                Unit_Normal_B       : out Unit_Vectors;
                Unit_Normal_C       : out Unit_Vectors;
                Unit_Tangent_B      : out Unit_Vectors;
                Unit_Tangent_C      : out Unit_Vectors;
                Segment_Bc_Distance : out Segment_Distances) is separate;

   procedure Update
               (Waypoint_C_Lat      : in      Earth_Positions;
                Waypoint_C_Long     : in      Earth_Positions;
                Unit_Normal_B       :     out Unit_Vectors;
                Unit_Normal_C       : in  out Unit_Vectors;
                Unit_Tangent_B      :     out Unit_Vectors;
                Unit_Tangent_C      : in  out Unit_Vectors;
                Segment_Bc_Distance :     out Segment_Distances) is separate;
```

```
end Steering_Vector_Operations;
```

```
separate (Waypoint_Steering.Steering_Vector_Operations)
procedure Initialize
             (Waypoint_A_Lat       : in  Earth_Positions;
              Waypoint_A_Long      : in  Earth_Positions;
              Waypoint_B_Lat       : in  Earth_Positions;
              Waypoint_B_Long      : in  Earth_Positions;
              Waypoint_C_Lat       : in  Earth_Positions;
              Waypoint_C_Long      : in  Earth_Positions;
              Unit_Normal_B        : out Unit_Vectors;
              Unit_Normal_C        : out Unit_Vectors;
              Unit_Tangent_B       : out Unit_Vectors;
              Unit_Tangent_C       : out Unit_Vectors;
              Segment_Bc_Distance : out Segment_Distances) is

-- ----------------------
-- --declaration section-
-- ----------------------

   Temp_Un_B     : Unit_Vectors;
   Temp_Un_C     : Unit_Vectors;
   Unit_Radial_A : Unit_Vectors;
   V_Length      : Sin_Cos_Ratio;

-----------------------------
--begin procedure Initialize-
-----------------------------

begin

-- --compute unit radial vectors

   Unit_Radial_A := U_R_Vector(Lat_Of_Point  => Waypoint_A_Lat,
                               Long_Of_Point => Waypoint_A_Long);
   Unit_Radial_B := U_R_Vector(Lat_Of_Point  => Waypoint_B_Lat,
                               Long_Of_Point => Waypoint_B_Long);
   Unit_Radial_C := U_R_Vector(Lat_Of_Point  => Waypoint_C_Lat,
                               Long_Of_Point => Waypoint_C_Long);

-- --compute UN_B

   Temp_Un_B      := Cross_Product(Left  => Unit_Radial_B,
                                   Right => Unit_Radial_A);
   V_Length       := Vector_Length(Temp_Un_B);
   Temp_Un_B      := Temp_Un_B / V_Length;
   Unit_Normal_B  := Temp_Un_B;

-- --compute UT_B

   Unit_Tangent_B := Cross_Product(Left  => Temp_Un_B,
                                   Right => Unit_Radial_B);

-- --compute UN_C and segment BC distance

   Compute_Segment_And_U_Nl_Vector
      (Unit_Radial1     => Unit_Radial_B,
       Unit_Radial2     => Unit_Radial_C,
       Unit_Normal2     => Temp_Un_C,
```

```
        Segment_Distance => Segment_Bc_Distance);

   Unit_Normal_C  := Temp_Un_C;

-- --compute UT_C

   Unit_Tangent_C := Cross_Product(Left  => Temp_Un_C,
                                   Right => Unit_Radial_C);

end Initialize;
```

```
separate (Waypoint_Steering.Steering_Vector_Operations)
procedure Update
             (Waypoint_C_Lat      : in     Earth_Positions;
              Waypoint_C_Long     : in     Earth_Positions;
              Unit_Normal_B       :    out Unit_Vectors;
              Unit_Normal_C       : in out Unit_Vectors;
              Unit_Tangent_B      :    out Unit_Vectors;
              Unit_Tangent_C      : in out Unit_Vectors;
              Segment_Bc_Distance :    out Segment_Distances) is
begin

-- --advance "C" vectors into "B" vectors

   Unit_Radial_B  := Unit_Radial_C;
   Unit_Normal_B  := Unit_Normal_C;
   Unit_Tangent_B := Unit_Tangent_C;

-- --calculate new values

   Unit_Radial_C  := U_R_Vector(Lat_Of_Point  => Waypoint_C_Lat,
                                Long_Of_Point => Waypoint_C_Long);

   Compute_Segment_And_U_Nl_Vector
      (Unit_Radial1     => Unit_Radial_B,
       Unit_Radial2     => Unit_Radial_C,
       Unit_Normal2     => Unit_Normal_C,
       Segment_Distance => Segment_Bc_Distance);

   Unit_Tangent_C := Cross_Product(Left  => Unit_Normal_C,
                                   Right => Unit_Radial_C);

end Update;
```

```
separate (Waypoint_Steering)
procedure Compute_Turn_Angle_And_Direction
             (Unit_Normal_C               : in     Unit_Vectors;
              Unit_Tangent_B              : in     Unit_Vectors;
              Unit_Tangent_C              : in     Unit_Vectors;
              Tan_Of_One_Half_Turn_Angle :    out Tan_Ratio;
              Turn_Direction              :    out Turning_Directions) is

-- ----------------------
-- --declaration section-
-- ----------------------

   Tan_Of_Half       : Tan_Ratio renames Tan_Of_One_Half_Turn_Angle;

   Cos_Of_Turn_Angle : Sin_Cos_Ratio;
   Sin_Of_Turn_Angle : Sin_Cos_Ratio;

-------------------------------------------------
--begin procedure Compute_Turn_Angle_and_Direction-
-------------------------------------------------

begin

   Cos_Of_Turn_Angle := Dot_Product(Left  => Unit_Tangent_B,
                                    Right => Unit_Tangent_C);
   Sin_Of_Turn_Angle := Dot_Product(Left  => Unit_Tangent_B,
                                    Right => Unit_Normal_C);

   Tan_Of_Half := abs(Sin_Of_Turn_Angle /
                      (Tan_Ratio(1.0) + Cos_Of_Turn_Angle));

   if Sin_Of_Turn_Angle < 0.0 then
      Turn_Direction := Left_Turn;
   else
      Turn_Direction := Right_Turn;
   end if;

end Compute_Turn_Angle_And_Direction;
```

```
separate (Waypoint_Steering)
package body Crosstrack_And_Heading_Error_Operations is

   procedure Compute_When_Turning
               (Distance_To_B       : in  Segment_Distances;
                Nonturning_Distance : in  Segment_Distances;
                Unit_Radial_M       : in  Unit_Vectors;
                Unit_Normal_B       : in  Unit_Vectors;
                Unit_Tangent_B      : in  Unit_Vectors;
                Turn_Direction      : in  Turning_Directions;
                Ground_Velocity     : in  Velocity_Vectors;
                Turn_Radius         : in  Segment_Distances;
                Crosstrack_Error    : out Segment_Distances;
                Heading_Error       : out Angles) is separate;

   procedure Compute_When_Not_Turning
               (Unit_Radial_M     : in  Unit_Vectors;
                Unit_Normal_B     : in  Unit_Vectors;
                Ground_Velocity   : in  Velocity_Vectors;
                Crosstrack_Error  : out Segment_Distances;
                Heading_Error     : out Angles) is separate;

   procedure Compute
               (Distance_To_B       : in  Segment_Distances;
                Nonturning_Distance : in  Segment_Distances;
                Unit_Radial_M       : in  Unit_Vectors;
                Unit_Normal_B       : in  Unit_Vectors;
                Unit_Tangent_B      : in  Unit_Vectors;
                Turn_Direction      : in  Turning_Directions;
                Turn_Status         : in  Turning_Statuses;
                Ground_Velocity     : in  Velocity_Vectors;
                Turn_Radius         : in  Segment_Distances;
                Crosstrack_Error    : out Segment_Distances;
                Heading_Error       : out Angles) is separate;

end Crosstrack_And_Heading_Error_Operations;
```

```
separate (Waypoint_Steering.Crosstrack_And_Heading_Error_Operations)
procedure Compute_When_Turning
             (Distance_To_B       : in  Segment_Distances;
              Nonturning_Distance : in  Segment_Distances;
              Unit_Radial_M       : in  Unit_Vectors;
              Unit_Normal_B       : in  Unit_Vectors;
              Unit_Tangent_B      : in  Unit_Vectors;
              Turn_Direction      : in  Turning_Directions;
              Ground_Velocity     : in  Velocity_Vectors;
              Turn_Radius         : in  Segment_Distances;
              Crosstrack_Error    : out Segment_Distances;
              Heading_Error       : out Angles) is

-- ----------------------
-- --declaration section-
-- ----------------------

   Crosstrack_Distance  : Segment_Distances;
   D_Y                  : Segment_Distances;
   D_X                  : Segment_Distances;
   D_H_D_C              : Segment_Distances;
   Direction            : INTEGER;
   Dot_Prod_Result      : Sin_Cos_Ratio;
   N_H_D_C              : Segment_Distances;
   R_M                  : Segment_Distances;

-------------------------------------
--begin procedure Compute_When_Turning-
-------------------------------------

begin

-- --convert turn direction to an integer value

   if Turn_Direction = Left_Turn then
      Direction := -1;
   else
      Direction := 1;
   end if;

-- --get the sine of the angle (which approximately equals the angle)
-- --between the actual and desired UR_M,
-- --and then compute crosstrack distance

   Dot_Prod_Result     := Dot_Product (Left  => Unit_Radial_M,
                                       Right => Unit_Normal_B);
   Crosstrack_Distance := Dot_Prod_Result * Earth_Radius;

-- --compute the radius of the circle that the missile is actually traversing

   D_Y := Turn_Radius - Crosstrack_Distance * Direction;
   D_X := Distance_To_B - Nonturning_Distance;
   R_M := Sqrt(D_X * D_X + D_Y * D_Y);

-- --compute crosstrack error

   Crosstrack_Error := (Turn_Radius - R_M) * Direction;
```

```
-- --compute heading error

   N_H_D_C := Direction * Unit_Tangent_B(Z) * D_X -
              Unit_Normal_B(Z) * D_Y;
   D_H_D_C := -DIrection * Unit_Normal_B(Z) * D_X -
              Unit_Tangent_B(Z) * D_Y;

   Heading_Error := Arctan((N_H_D_C * Ground_Velocity(N) -
                            D_H_D_C * Ground_Velocity(E)) /
                           (D_H_D_C * Ground_Velocity(N) +
                            N_H_D_C * Ground_Velocity(E)));

end Compute_When_Turning;
```

```
separate (Waypoint_Steering.Crosstrack_And_Heading_Error_Operations)
procedure Compute_When_Not_Turning
             (Unit_Radial_M    : in  Unit_Vectors;
              Unit_Normal_B    : in  Unit_Vectors;
              Ground_Velocity  : in  Velocity_Vectors;
              Crosstrack_Error : out Segment_Distances;
              Heading_Error    : out Angles) is

-- ---------------------
-- --local declarations-
-- ---------------------

   Dot_Prod_Result : Sin_Cos_Ratio;
   D_H_D_C         : Sin_Cos_Ratio;
   N_H_D_C         : Sin_Cos_Ratio;

-----------------------------------------
--begin procedure Compute_When_Not_Turning-
-----------------------------------------

begin

-- --get the sine of the angle (which approximately equals the angle)
-- --between the actual and desired UR_M,
-- --and then compute crosstrack distance/error

   Dot_Prod_Result  := Dot_Product (Left  => Unit_Radial_M,
                                   Right => Unit_Normal_B);
   Crosstrack_Error := Dot_Prod_Result * Earth_Radius;

-- --compute heading error

   N_H_D_C := - Unit_Normal_B(Z);
   D_H_D_C := Unit_Normal_B(Y) * Unit_Radial_M(X) -
              Unit_Normal_B(X) * Unit_Radial_M(Y);

   Heading_Error := Arctan((N_H_D_C * Ground_Velocity(N) -
                            D_H_D_C * Ground_Velocity(E)) /
                           (D_H_D_C * Ground_Velocity(N) +
                            N_H_D_C * Ground_Velocity(E)));

end Compute_When_Not_Turning;
```

```
separate (Waypoint_Steering.Crosstrack_And_Heading_Error_Operations)
procedure Compute
             (Distance_To_B       : in  Segment_Distances;
              Nonturning_Distance : in  Segment_Distances;
              Unit_Radial_M       : in  Unit_Vectors;
              Unit_Normal_B       : in  Unit_Vectors;
              Unit_Tangent_B      : in  Unit_Vectors;
              Turn_Direction      : in  Turning_Directions;
              Turn_Status         : in  Turning_Statuses;
              Ground_Velocity     : in  Velocity_Vectors;
              Turn_Radius         : in  Segment_Distances;
              Crosstrack_Error    : out Segment_Distances;
              Heading_Error       : out Angles) is
begin
   if Turn_Status = Turning then
      Compute_When_Turning
                  (Distance_To_B       => Distance_To_B       ,
                   Nonturning_Distance => Nonturning_Distance ,
                   Unit_Radial_M       => Unit_Radial_M       ,
                   Unit_Normal_B       => Unit_Normal_B       ,
                   Unit_Tangent_B      => Unit_Tangent_B      ,
                   Turn_Direction      => Turn_Direction      ,
                   Ground_Velocity     => Ground_Velocity     ,
                   Turn_Radius         => Turn_Radius         ,
                   Crosstrack_Error    => Crosstrack_Error    ,
                   Heading_Error       => Heading_Error       );
   else
      Compute_When_Not_Turning
                  (Unit_Radial_M    => Unit_Radial_M    ,
                   Unit_Normal_B    => Unit_Normal_B    ,
                   Ground_Velocity  => Ground_Velocity  ,
                   Crosstrack_Error => Crosstrack_Error ,
                   Heading_Error    => Heading_Error    );
   end if;
end Compute;
```

```
separate (Waypoint_Steering)
function Distance_To_Current_Waypoint
            (Unit_Radial_M  : Unit_Vectors;
             Unit_Tangent_B : Unit_Vectors) return Segment_Distances is

-- ---------------------
-- --local declarations-
-- ---------------------

   Dot_Prod_Result : Sin_Cos_Ratio;

-----------------------------------------------
--begin function Distance_To_Current_Waypoint-
-----------------------------------------------

begin

-- --dot product of UR_M and UT_B, because of the geometry, equals
-- --the angle between UR_M and UR_B
   Dot_Prod_Result := Dot_Product (Left  => Unit_Radial_M,
                                   Right => Unit_Tangent_B);

   return Dot_Prod_Result * Earth_Radius;
end Distance_To_Current_Waypoint;
```

```
separate (Waypoint_Steering)
procedure Compute_Turning_And_Nonturning_Distances
             (Tan_Of_One_Half_Turn_Angle : in     Tan_Ratio;
              Segment_Bc_Distance        : in     Distances;
              Turn_Radius                : in     Distances;
              Turning_Distance           :    out Distances;
              Nonturning_Distance        :    out Distances) is

-- ----------------------
-- --declaration section-
-- ----------------------

   Temp_Turning_Distance : Distances;

-------------------------------------------------------------
--begin procedure Compute_Turning_And_Nonturning_Distances-
-------------------------------------------------------------

begin

   Temp_Turning_Distance := Turn_Radius * Tan_Of_One_Half_Turn_Angle;
   Turning_Distance      := Temp_Turning_Distance;
   Nonturning_Distance   := Segment_Bc_Distance -  Temp_Turning_Distance;

end Compute_Turning_And_Nonturning_Distances;
```

```
separate (Waypoint_Steering)
package body Turn_Test_Operations is

   function Stop_Test
               (Distance_To_B       : Distances;
                Nonturning_Distance : Distances;
                Lead_Distance       : Distances)
               return Turning_Statuses is separate;

   function Start_Test
               (Distance_To_B       : Distances;
                Turning_Distance    : Distances;
                Lead_Distance       : Distances)
               return Turning_Statuses is separate;

end Turn_Test_Operations;
```

```
separate (Waypoint_Steering.Turn_Test_Operations)
function Stop_Test
            (Distance_To_B       : Distances;
             Nonturning_Distance : Distances;
             Lead_Distance       : Distances) return Turning_Statuses is

-- ----------------------
-- --declaration section-
-- ----------------------

   Turn_Status : Turning_Statuses := Turning;

----------------------------
--begin function Stop_Test-
----------------------------

begin

   if Distance_To_B <= (Lead_Distance + Nonturning_Distance) then
      Turn_Status := Not_Turning;
   end if;

   return Turn_Status;

end Stop_Test;
```

```
separate (Waypoint_Steering.Turn_Test_Operations)
function Start_Test
            (Distance_To_B       : Distances;
             Turning_Distance    : Distances;
             Lead_Distance       : Distances) return Turning_Statuses is

-- ----------------------
-- --declaration section-
-- ----------------------

   Turn_Status : Turning_Statuses := Not_Turning;

----------------------------
--begin function Start_Test-
----------------------------

begin

   if Distance_To_B <= (Lead_Distance + Turning_Distance) then
      Turn_Status := Turning;
   end if;

   return Turn_Status;

end Start_Test;
```

```
with Geometric_Operations;
separate (Waypoint_Steering)
package body Steering_Vector_Operations_With_Arcsin is

   package Geo renames Geometric_Operations;

-- ----------------------------
-- --instantiate required parts-
-- ----------------------------

   function U_R_Vector is new
              Geo.Unit_Radial_Vector (Indices          => Indices,
                                      Earth_Positions => Earth_Positions,
                                      Sin_Cos_Ratio   => Sin_Cos_Ratio,
                                      Unit_Vectors    => Unit_Vectors);

   procedure Compute_Segment_And_U_Nl_Vector is new
                Geo.Compute_Segment_And_Unit_Normal_Vector_With_Arcsin
                   (Indices           => Indices,
                    Earth_Distances   => Earth_Distances,
                    Segment_Distances => Segment_Distances,
                    Radians           => Radians,
                    Sin_Cos_Ratio     => Sin_Cos_Ratio,
                    Unit_Vectors      => Unit_Vectors,
                    Earth_Radius      => Earth_Radius);

-- ---------------------
-- --local declarations-
-- ---------------------

   Unit_Radial_B : Unit_Vectors;
   Unit_Radial_C : Unit_Vectors;

pragma PAGE;
   procedure Initialize
               (Waypoint_A_Lat      : in  Earth_Positions;
                Waypoint_A_Long     : in  Earth_Positions;
                Waypoint_B_Lat      : in  Earth_Positions;
                Waypoint_B_Long     : in  Earth_Positions;
                Waypoint_C_Lat      : in  Earth_Positions;
                Waypoint_C_Long     : in  Earth_Positions;
                Unit_Normal_B       : out Unit_Vectors;
                Unit_Normal_C       : out Unit_Vectors;
                Unit_Tangent_B      : out Unit_Vectors;
                Unit_Tangent_C      : out Unit_Vectors;
                Segment_Bc_Distance : out Segment_Distances) is

--     ----------------------
--     --declaration section-
--     ----------------------

       Temp_Un_B     : Unit_Vectors;
       Temp_Un_C     : Unit_Vectors;
       Unit_Radial_A : Unit_Vectors;
       V_Length      : Sin_Cos_Ratio;

--    ----------------------------
```

```
--    --begin procedure Initialize-
--    ------------------------------

   begin

--      --compute unit radial vectors

        Unit_Radial_A := U_R_Vector(Lat_Of_Point  => Waypoint_A_Lat,
                                    Long_Of_Point => Waypoint_A_Long);
        Unit_Radial_B := U_R_Vector(Lat_Of_Point  => Waypoint_B_Lat,
                                    Long_Of_Point => Waypoint_B_Long);
        Unit_Radial_C := U_R_Vector(Lat_Of_Point  => Waypoint_C_Lat,
                                    Long_Of_Point => Waypoint_C_Long);

--      --compute UN_B

        Temp_Un_B      := Cross_Product(Left  => Unit_Radial_B,
                                        Right => Unit_Radial_A);
        V_Length       := Vector_Length(Temp_Un_B);
        Temp_Un_B      := Temp_Un_B / V_Length;
        Unit_Normal_B  := Temp_Un_B;

--      --compute UT_B

        Unit_Tangent_B := Cross_Product(Left  => Temp_Un_B,
                                        Right => Unit_Radial_B);

--      --compute UN_C and segment BC distance

        Compute_Segment_And_U_Nl_Vector
           (Unit_Radial1     => Unit_Radial_B,
            Unit_Radial2     => Unit_Radial_C,
            Unit_Normal2     => Temp_Un_C,
            Segment_Distance => Segment_Bc_Distance);

        Unit_Normal_C  := Temp_Un_C;

--      --compute UT_C

        Unit_Tangent_C := Cross_Product(Left  => Temp_Un_C,
                                        Right => Unit_Radial_C);

   end Initialize;

pragma PAGE;
   procedure Update
                (Waypoint_C_Lat      : in     Earth_Positions;
                 Waypoint_C_Long     : in     Earth_Positions;
                 Unit_Normal_B       :    out Unit_Vectors;
                 Unit_Normal_C       : in out Unit_Vectors;
                 Unit_Tangent_B      :    out Unit_Vectors;
                 Unit_Tangent_C      : in out Unit_Vectors;
                 Segment_Bc_Distance :    out Segment_Distances) is
   begin

--      --advance "C" vectors into "B" vectors
```

```
      Unit_Radial_B  := Unit_Radial_C;
      Unit_Normal_B  := Unit_Normal_C;
      Unit_Tangent_B := Unit_Tangent_C;

--    --calculate new values

      Unit_Radial_C  := U_R_Vector(Lat_Of_Point  => Waypoint_C_Lat,
                                   Long_Of_Point => Waypoint_C_Long);

      Compute_Segment_And_U_Nl_Vector
         (Unit_Radial1     => Unit_Radial_B,
          Unit_Radial2     => Unit_Radial_C,
          Unit_Normal2     => Unit_Normal_C,
          Segment_Distance => Segment_Bc_Distance);

      Unit_Tangent_C := Cross_Product(Left  => Unit_Normal_C,
                                      Right => Unit_Radial_C);

   end Update;

end Steering_Vector_Operations_With_Arcsin;
```

```
separate (Waypoint_Steering)
function Distance_To_Current_Waypoint_With_Arcsin
            (Unit_Radial_M  : Unit_Vectors;
             Unit_Tangent_B : Unit_Vectors) return Segment_Distances is

-- ---------------------
-- --local declarations-
-- ---------------------

   Dot_Prod_Result : Sin_Cos_Ratio;

-----------------------------------------------
--begin function Distance_To_Curr...._Waypoint-
-----------------------------------------------

begin

-- --dot product of UR_M and UT_B, because of the geometry, equals
-- --the angle between UR_M and UR_B
   Dot_Prod_Result := Dot_Product (Left  => Unit_Radial_M,
                                   Right => Unit_Tangent_B);

   return Arcsin(Dot_Prod_Result) * Earth_Radius;

end Distance_To_Current_Waypoint_With_Arcsin;
```

(This page left intentionally blank.)

3.3.4.2 AUTOPILOT (PACKAGE BODY) TLCSC (CATALOG #P305-0)

This package body contains bodies for the the three packages nested in the Autopilot package. Each of these packages is separately compiled.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.4.2.1 REQUIREMENTS ALLOCATION

This part meets the following CAMP requirements.

Name	Type	Req. Allocation
Integral Plus Proportional	generic package	R048
Pitch autopilot	generic package	R059
Lateral/Directional autopilot	generic package	R064

3.3.4.2.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.3 INPUT/OUTPUT

None.

3.3.4.2.4 LOCAL DATA

None.

3.3.4.2.5 PROCESS CONTROL

Not applicable.

3.3.4.2.6 PROCESSING

The following describes the processing performed by this part:

```
package body Autopilot is

   package body Integral_Plus_Proportional_Gain is separate;

   package body Pitch_Autopilot is separate;

   package body Lateral_Directional_Autopilot is separate;
```

```
end Autopilot;
```

3.3.4.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.4.2.8 LIMITATIONS

None.

3.3.4.2.9 LLCSC DESIGN

3.3.4.2.9.1 INTEGRAL_PLUS_PROPORTIONAL_GAIN (PACKAGE BODY) PACKAGE DESIGN (CATALOG #P306-0)

This part contains subprograms to implement the calculations and logic necessary to implement an integral plus proportional gain control loop. It also contains a subprogram to update the value for the proportional gain.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.4.2.9.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R048.

3.3.4.2.9.1.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Input_Signals	generic float	Type of values input to part
Gains	generic float	Type of gain applied to input
Integrated_Signals	generic float	Input signal put through integrator

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Description
Initial_Proportional_Gain	Gains	Initial value of proportional gain applied to input signal

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Overloads Input_Signals * Gains return Integrated_Signals for proportional gain
Tustin_Integrate	function	Performs Tustin integrator with limit

3.3.4.2.9.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Proportional_Gain	Gains	Gain applied to input signal.

3.3.4.2.9.1.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.1.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Autopilot)
package body Integral_Plus_Proportional_Gain is

   Proportional_Gain: Gains := Initial_Proportional_Gain;

end Integral_Plus_Proportional_Gain;
```

3.3.4.2.9.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.4.2.9.1.8 LIMITATIONS

None.

3.3.4.2.9.1.9 LLCSC DESIGN

None.

3.3.4.2.9.1.10 UNIT DESIGN

3.3.4.2.9.1.10.1 INTEGRATE UNIT DESIGN

This function performs the integral plus proportional gain function. The logic to perform this operation is as follows:

Accept new input signal Use Tustin Integrator with Limit to adjust input signal (If Integrator state within limit do not change input. If Integrator state at positive limit and input signal > 0 then set input to 0. If Integrator state at negative limit and input signal < 0 then set input to 0.) Use Tustin Integrator to perform integration and limit functions. Output = Proportional_-Gain * input signal + Tustin Integrator State.

3.3.4.2.9.1.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R048 (2).

3.3.4.2.9.1.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.1.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table summarizes the formal parameters required by this part:

Name	Type	Description
Signal	Input_Signals	Value of Input_Signal for integral plus proportional gain.

3.3.4.2.9.1.10.1.4 LOCAL DATA

None.

3.3.4.2.9.1.10.1.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.1.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
function Integrate (Signal: Input_Signals) return Integrated_Signals is
begin
   return (Tustin_Integrate (Signal) + Signal * Proportional_Gain);
end Integrate;
```

3.3.4.2.9.1.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top-level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top-level component:

Name	Type	Source	Description
"*"	function	Generic Fml Subp	Overloads Input_Signals * Gains return Integrated_Signals for proportional gain
Tustin_ Integrate	function	Generic Fml Subp	Performs Tustin integrator with limit

Data types:

The following table summarizes the types required by this part and defined elsewhere in the parent top-level component:

Name	Type	Source	Description
Input_Signals	generic float	Generic fml type	Type of values input to part
Gains	generic float	Generic fml type	Type of gain applied to input
Integrated_Signals	generic float	Generic fml type	Input signal put through integrator

Data objects:

The following table summarizes the objects required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Proportional_Gain	Gains	Package Body	Gain applied to input signal

3.3.4.2.9.1.10.1.8 LIMITATIONS

None.

3.3.4.2.9.1.10.2 UPDATE_PROPORTIONAL_GAIN UNIT DESIGN

This procedure updates the value stored for the proportional gain.

3.3.4.2.9.1.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R048.

3.3.4.2.9.1.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.1.10.2.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Gains	generic float	Type of gain applied to input

FORMAL PARAMETERS:

The following table summarizes the formal parameters required by this part:

Name	Type	Description
New_Proportional_Gain	Gains	Value to update proportional gain

3.3.4.2.9.1.10.2.4 LOCAL DATA

None.

3.3.4.2.9.1.10.2.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.1.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Proportional_Gain (New_Proportional_Gain : in Gains) is

begin

  Proportional_Gain := New_Proportional_Gain;

end Update_Proportional_Gain;
```

3.3.4.2.9.1.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

Data objects:

The following table summarizes the objects required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Proportional_Gain	Gains	Package Body	Gain applied to input signal

3.3.4.2.9.1.10.2.8 LIMITATIONS

None.

3.3.4.2.9.2 LATERAL DIRECTIONAL AUTOPILOT (PACKAGE BODY) PACKAGE DESIGN (CATALOG #P308-0)

This package body implements the Lateral Directional Autopilot function. It contains the instantiation of the Integral Plus Proportional Gain packages for the integrator loops of both the Roll Command Error and the Lateral Acceleration feedback, as well as subprogram bodies for operations declared in the package specification.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.4.2.9.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R064.

3.3.4.2.9.2.2 LOCAL ENTITIES DESIGN

Packages:

Instantiates Integral plus proportional gain package for aileron roll command and for filtered lateral directional acceleration. Also instantiates Tustin integrator to implement each of the integral plus proportional gain packages.

3.3.4.2.9.2.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Roll_Commands	Generic Float	Type for input commands from user program
Roll_Attitudes	Generic Float	Type for measure missile roll attitude
Roll_Command_Gains	Generic Float	Gain to Roll commands in integrator loop
Missile_Accelerations	Generic Float	Type for measured lateral acceleration
Acceleration_Gains	Generic Float	Proportional gain applied measured acceleration
Rudder_Cmd_Roll_Rate_Gains	Generic Float	Gain applied to roll rate feedback for rudder cmd
Gravitational_Accelerations	Generic Float	Type for measured gravitational acceleration
Velocities	Generic Float	Type for measured missile velocity
Trig_Value	Generic Float	Type for result of sin function
Fin_Deflections	Generic Float	Type for rudder and aileron commands
Feedback_Rates	Generic Float	Type for measured roll and yaw rates
Feedback_Rate_Gains	Generic Float	Gain applied to yaw rate feedback

Data objects:

The following table summarizes the generic formal objects required by this part:

Name	Type	Description
Initial_Aileron_Integrator_Gain	Roll_Command_Gains	Gain used to initialize aileron loop
Initial_Aileron_Integrator_Limit	Fin_Deflections	Initial limit on aileron integrator output
Initial_Roll_Command_Proportional_Gain	Roll_Command_Gains	Gain used to initialize integrator plus proportional gain loop
Initial_Roll_Rate_Gain_For_Aileron	Feedback_Rate_Gains	Gain to measured roll rate for aileron cmd
Initial_Yaw_Rate_Gain_For_Aileron	Feedback_Rate_Gains	Gain to measured yaw rate for aileron cmd
Initial_Rudder_Integrator_Gain	Acceleration_Gains	Initial gain in rudder integrator loop
Initial_Rudder_Integrator_Limit	Fin_Deflections	Initial limit on rudder integrator output
Initial_Yaw_Rate_Gain_For_Rudder	Feedback_Rate_Gains	Gain to measured yaw for rudder command
Initial_Roll_Rate_Gain_For_Rudder	Rudder_Cmd_Roll_Rate_Gains	Gain to measured roll rate for rudder cmd
Initial_Acceleration_Proportional_Gain	Acceleration_Gains	Initial Prop. gain in integrator plus proportional gain loop

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
Aileron control loop limiters and filters		
Roll_Error_Limit	function	Limiter for roll error
Aileron_Command_Limit	function	Limit on command signal to aileron
Roll_Command_Filter	function	Filter applied to input roll command
Rudder control loop limiters, filters, and operations		
Rudder_Command_Limit	function	Limit on command signal to rudder
Yaw_Rate_Filter	function	Filter applied to measured yaw rate
Acceleration_Filter	function	Filter applied to measured acceleration feedback
Sin	function	Sin function applied to measured roll attitude
Aileron control loop gain and updater functions		
"-"	function	Subtracts Roll_Attitudes from Roll_Commands returning Roll_Error
"*"	function	Multiplies Roll_Commands by Roll_Command_Gains for input to Aileron integrator
"*"	function	Multiplies Feedback_Rates for measured roll rate by Feedback_Rate_Gains for Fin_Deflections
Rudder control loop gain and updater functions		
"*"	function	Multiplies Missile_Accelerations by Acceleration_Gains returns Fin_Deflections for proportional loop of integral plus proportional gain
"*"	function	Multiplies Feedback_Rates by Rudder_Cmd_Roll_Rate_Gains returns Feedback_Rates
"*"	function	Multiplies Gravitational_Accelerations by Trig_Value returns Gravitational_accelerations
"/"	function	Divides Gravitational_Accelerations by Velocities returns Feedback_Rates

3.3.4.2.9.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Objects for Aileron Control Loop		
Aileron_Cmd_Roll_Rate_Gain	Feedback_Rate_Gains	Gain on roll rate feedback in aileron command loop
Aileron_Cmd_Yaw_Rate_Gain	Feedback_Rate_Gains	Gain on yaw rate feedback in aileron command loop
Objects for rudder control loop		
Rudder_Cmd_Roll_Rate_Gain	Rudder_Cmd_Roll_Rate_Gains	Gain to roll rate feedback in rudder command loop
Rudder_Cmd_Feedback_Rate_Gain	Feedback_Rate_Gains	Gain to yaw rate feedback in rudder command loop

3.3.4.2.9.2.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.2.6 PROCESSING

The following describes the processing performed by this part:

```
with Signal_Processing;
separate (Autopilot)
package body Lateral_Directional_Autopilot is

-- --Initial values for Aileron Control Loop

   Aileron_Cmd_Roll_Rate_Gain     : Feedback_Rate_Gains :=
                                       Initial_Roll_Rate_Gain_For_Aileron;
   Aileron_Cmd_Yaw_Rate_Gain      : Feedback_Rate_Gains :=
                                       Initial_Yaw_Rate_Gain_For_Aileron;

-- --Initial values for rudder control loop

   Rudder_Cmd_Roll_Rate_Gain      : Rudder_Cmd_Roll_Rate_Gains :=
                                       Initial_Roll_Rate_Gain_For_Rudder;
   Rudder_Cmd_Feedback_Rate_Gain  : Feedback_Rate_Gains :=
                                       Initial_Yaw_Rate_Gain_For_Rudder;

-- Packages for Aeliron control loop

   package Aileron_Cmd_Tustin_Integrator is new
             Signal_Processing.Tustin_Integrator_With_Limit
                (Signals                => Roll_Commands,
```

```
                States                => Fin_Deflections,
                Gained_Signals        => Fin_Deflections,
                Gains                 => Roll_Command_Gains,
                Initial_Tustin_Gain   => Initial_Aileron_Integrator_Gain,
                Initial_Signal_Level  => 0.0,
                Initial_Signal_Limit  => Initial_Aileron_Integrator_Limit);

   package Aileron_Cmd_Integral_Plus_Proportional_Gain is new
      Integral_Plus_Proportional_Gain
         (Input_Signals             => Roll_Commands,
          Gains                     => Roll_Command_Gains,
          Integrated_Signals        => Fin_Deflections,
          Initial_Proportional_Gain => Initial_Roll_Command_Proportional_Gain,
          Tustin_Integrate          => Aileron_Cmd_Tustin_Integrator.Integrate);

   package Aileron_Loop renames Aileron_Cmd_Integral_Plus_Proportional_Gain;

-- Packages for rudder control loop

   package Rudder_Cmd_Tustin_Integrator is new
            Signal_Processing.Tustin_Integrator_With_Limit
               (Signals              => Missile_Accelerations,
                States               => Fin_Deflections,
                Gained_Signals       => Fin_Deflections,
                Gains                => Acceleration_Gains,
                Initial_Tustin_Gain  => Initial_Rudder_Integrator_Gain,
                Initial_Signal_Level => 0.0,
                Initial_Signal_Limit => Initial_Rudder_Integrator_Limit);

   package Rudder_Cmd_Integral_Plus_Proportional_Gain is new
      Integral_Plus_Proportional_Gain
         (Input_Signals             => Missile_Accelerations,
          Gains                     => Acceleration_Gains,
          Integrated_Signals        => Fin_Deflections,
          Initial_Proportional_Gain => Initial_Acceleration_Proportional_Gain,
          Tustin_Integrate          => Rudder_Cmd_Tustin_Integrator.Integrate);

   package Rudder_Loop renames Rudder_Cmd_Integral_Plus_Proportional_Gain;

end Lateral_Directional_Autopilot;
```

3.3.4.2.9.2.7 UTILIZATION OF OTHER ELEMENTS

The following library units are with'd by this part:

1. Signal_Processing

UTILIZATION OF EXTERNAL ELEMENTS:

Packages:

The following table summarizes the external packages required by this part:

Name	Type	Source	Description
Tustin_Integrator_With_Limit	Generic Package	1.	This package is required for the integration function in the integral plus proportional packages. It is instantiated for the roll command control loop and for the acceleration feedback control loop.

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Packages:

The following table summarizes the packages required by this part but defined elsewhere in the parent top level component:

Name	Type	Source	Description
Integral_Plus_Proportional_Gain	Generic Package	Package Spec.	Performs integrator function on roll command error and lateral acceleration feedback

Data types:

The following table summarizes the types required by this part and defined in ancestral units:

Name	Type	Source	Description
Aileron_Rudder_Commands	Record	Package Spec.	Defines record with components for rud- and aileron commands

3.3.4.2.9.2.8 LIMITATIONS

None.

3.3.4.2.9.2.9 LLCSC DESIGN

None.

3.3.4.2.9.2.10 UNIT DESIGN

3.3.4.2.9.2.10.1 INITIALIZE_LATERAL_DIRECTIONAL_AUTOPILOT UNIT DESIGN

Initializes state of integrator in lateral directional autopilot.

3.3.4.2.9.2.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R064 (2).

3.3.4.2.9.2.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.2.10.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Fin_Deflections	Generic Float	Type for rudder and aileron commands
Missile_Accelerations	Generic Float	Type for measured lateral acceleration
Roll_Commands	Generic Float	Type for input commands from user program
Feedback_Rates	Generic Float	Type for measured roll and yaw rates
Velocities	Generic Float	Type for measured missile velocity
Roll_Attitudes	Generic Float	Type for measure missile roll attitude

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Name	Type	Description
For Aileron State Initialization		
"-"	function	Subtracts Roll_Attitudes from Roll_Commands returning Roll_Error
"*"	function	Multiplies Roll_Commands by Roll_Command_Gains for input to Aileron integrator
"*"	function	Multiplies Feedback_Rates for measured roll and yaw rates by Feedback_Rate_Gains for Fin_Deflections
For Aileron State Initialization		
"*"	function	Multiplies Gravitational Accelerations by Trig_Value returns Gravitational_Accelerations
"/"	function	Divides Gravitational_Accelerations by Velocities returns Feedback_Rates
"*"	function	Multiplies Feedback_Rates by Rudder_Cmd_Roll_Rate_Gains returns Feedback_Rates
"*"	function	Multiplies Missile_Accelerations by Acceleration_Gains returns Fin_Deflections for proportional loop of integral plus proportional gain

FORMAL PARAMETERS:

The following table summarizes the formal parameters required by this part:

Name	Type	Description
Initial_Aileron_Command	Fin_Deflections	Initial state for aileron deflection
Initial_Rudder_Command	Fin_Deflections	Initial state for rudder deflection
Gravitational_Acceleration	Gravitational_Accelerations	Measured gravitational acceleration from NAV
Roll_Command	Roll_Commands	Initial roll command
Roll_Attitude	Roll_Attitudes	Measured roll attitude
Roll_Rate	Feedback_Rates	Measured roll rate feedback
Yaw_Rate	Feedback_Rates	Measured yaw rate feedback
Missile_Velocity	Velocities	Measured velocity from NAV
Lateral_Acceleration	Missile_Accelerations	Measured lateral accel- from NAV

3.3.4.2.9.2.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Gained_Roll_Command_Signal	Fin_Deflections	Input to integrator
Initial_Aileron_State	Fin_Delfections	Initial state of integrator for aileron
Initial_Rudder_State	Fin_Deflections	Initial state of integrator for rudder

3.3.4.2.9.2.10.1.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.2.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Initialize_Lateral_Directional_Autopilot
            (Initial_Aileron_Command    : in Fin_Deflections;
             Initial_Rudder_Command     : in Fin_Deflections;
             Gravitational_Acceleration : in Gravitational_Accelerations;
             Roll_Command               : in Roll_Commands;
             Roll_Attitude              : in Roll_Attitudes;
             Roll_Rate                  : in Feedback_Rates;
             Yaw_Rate                   : in Feedback_Rates;
```

```
                Missile_Velocity          : in Velocities;
                Lateral_Acceleration      : in Missile_Accelerations) is

      Gained_Roll_Command_Signal : Fin_Deflections;

      Initial_Aileron_State : Fin_Deflections;

      Initial_Rudder_State : Fin_Deflections;

   begin

      Gained_Roll_Command_Signal :=
               (Roll_Command - Roll_Attitude) *
               Initial_Roll_Command_Proportional_Gain;

      Initial_Aileron_State :=
                Initial_Aileron_Command -
                Gained_Roll_Command_Signal +
                Yaw_Rate * Aileron_Cmd_Yaw_Rate_Gain +
                Roll_Rate * Aileron_Cmd_Yaw_Rate_Gain;

      Initial_Rudder_State :=
                Initial_Rudder_Command -
                (Yaw_Rate -
                 (Gravitational_Acceleration * Sin (Roll_Attitude)
                          / Missile_Velocity) -
                 (Roll_Rate * Rudder_Cmd_Roll_Rate_Gain)
                ) * Rudder_Cmd_Feedback_Rate_Gain -
                Lateral_Acceleration * Initial_Acceleration_Proportional_Gain;

      Aileron_Cmd_Tustin_Integrator.Reset
         (Integrator_State => Initial_Aileron_State,
          Signal           => 0.0);

      Rudder_Cmd_Tustin_Integrator.Reset
         (Integrator_State => Initial_Rudder_State,
          Signal           => 0.0);

    end Initialize_Lateral_Directional_Autopilot;
```

3.3.4.2.9.2.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Packages:

The following table summarizes the packages required by this part but defined elsewhere in the parent top level component:

Name	Type	Source	Description
Aileron_Cmd_Tustin_Integrator	Package	Package Body	Implements integrator for roll command loop
Rudder_Cmd_Tustin_Integrator	Package	Package Body	Implements integrator for acceleration feedback loop

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Reset	Function	Aileron_Cmd_Tustin_Integrator (Pack.)	Reinitializes state of integrator
Reset	Function	Rudder_Cmd_Tustin_Integrator (Pack.)	Reinitializes state of integrator

Data objects:

The following table summarizes the objects required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Aileron_Integrator_Gain	Roll_Command_Gains	Body	Gain on input to aileron command loop integrator
Aileron_Cmd_Roll_Rate_Gain	Feedback_Rate_Gains	Body	Gain on roll rate feedback in aileron command loop
Aileron_Cmd_Yaw_Rate_Gain	Feedback_Rate_Gains	Body	Gain on yaw rate feedback in aileron command loop
Rudder_Cmd_Roll_Rate_Gain	Rudder_Cmd_Roll_Rate_Gains	Body	Gain to roll rate feedback in rudder command loop
Rudder_Cmd_Feedback_Rate_Gain	Feedback_Rate_Gains	Body	Gain to yaw rate feedback in rudder command loop
Acceleration_Proportional_Gain	Acceleration_Gains	Body	Proportional gain to acceleration feedback in
Rudder_Integrator_Gain	Acceleration_Gains	Body	Gain on acceleration input to rudder command loop integrator

3.3.4.2.9.2.10.1.8 LIMITATIONS

None.

3.3.4.2.9.2.10.2 COMPUTE_AILERON_RUDDER_COMMANDS(FUNCTION BODY) UNIT DESIGN

Computes Aileron and Rudder commands based on roll command input and current missile state.

3.3.4.2.9.2.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R064 (3).

3.3.4.2.9.2.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.2.10.2.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the external types required by this part:

Name	Type	Description
Roll_Commands	Generic Float	Type for input commands from user program
Roll_Attitudes	Generic Float	Type for measure missile roll attitude
Feedback_Rates	Generic Float	Type for measured roll and yaw rates
Missile_Acclerations	Generic Float	Type for measured lateral acceleration
Fin_Deflections	Generic Float	Type for rudder and aileron commands
Velocities	Generic Float	Type for measured missile velocity
Gravitational_ Accelerations	Generic Float	Type for measured gravitationll acceleration

Subprograms:

The following table summarizes the generic formal subroutines required by this part:

Name	Type	Description
Aileron control loop limiters and filters		
Roll_Error_Limit	function	Limiter for roll error
Aileron_Command_Limit	function	Limit on command signal to aileron
Roll_Command_Filter	function	Filter applied to input roll command
Rudder control loop limiters, filters, and operations		
Rudder_Command_Limit	function	Limit on command signal to rudder
Sin	function	Sin function applied to measured roll attitude
Yaw_Rate_Filter	function	Filter applied to measured yaw rate
Acceleration_Filter	function	Filter applied to measured acceleration feedback
Aileron control loop gain and updater functions		
"-"	function	Subtracts Roll_Attitudes from Roll_Commands returning Roll_Error
"*"	function	Multiplies Feedback_Rates for measured roll and yaw rate by Feedback_Rate_Gains for Fin_Deflections
Rudder control loop gain and updater functions		
"*"	function	Multiplies Gravitational Accelerations by Trig_Value returns Gravitational_Accelerations
"/"	function	Divides Gravitational_Accelerations by Velocities returns Feedback_Rates
"*"	function	Multiplies Feedback_Rates by Rudder_Cmd_Roll_Rate_Gains returns Feedback_Rates

FORMAL PARAMETERS:

The following table summarizes the formal parameters required by this part:

Name	Type	Description
Roll_Command	Roll_Commands	Input roll command
Roll_Attitude	Roll_Attitudes	Measured roll attitude
Roll_Rate	Feedback_Rates	Measured roll rate feedback
Yaw_Rate	Feedback_Rates	Measured yaw rate feedback
Lateral_Acceleration	Missile_Accelerations	Measured lateral acceleration from NAV
Missile_Velocity	Velocities	Measured velocity from NAV
Gravitational_Acceleration	Gravitational_Accelerations	Measured gravitational acceleration from NAV

3.3.4.2.9.2.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Filtered_Roll_Command	Roll_Commands	Input roll command after filtering
Roll_Error	Roll_Commands	-Filtered_Roll_Command - Roll_Attitude (Limited)
Aileron_Integral_Output	Fin_Deflections	Output from aileron loop integral plus prop. gain
Filtered_Yaw_Rate	Feedback_Rates	Yaw rate feedback after filtering
Filtered_Lateral_Acceleration	Missile_Accelerations	Acceleration feedback after filtering
Rudder_Integral_Output	Fin_Deflections	Output from rudder loop integral plus prop. gain
Fin_Command	Aileron_Rudder_Commands	Contains aileron and rudder command components

3.3.4.2.9.2.10.2.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.2.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
function Compute_Aileron_Rudder_Commands
            (Roll_Command              : in Roll_Commands;
             Roll_Attitude             : in Roll_Attitudes;
             Roll_Rate                 : in Feedback_Rates;
```

```
                Yaw_Rate                    : in Feedback_Rates;
                Lateral_Acceleration        : in Missile_Accelerations;
                Missile_Velocity            : in Velocities;
                Gravitational_Acceleration  : in Gravitational_Accelerations)
          return Aileron_Rudder_Commands is

      Filtered_Roll_Command        : Roll_Commands;
      Roll_Error                   : Roll_Commands;
      Aileron_Integral_Output      : Fin_Deflections;
      Filtered_Yaw_Rate            : Feedback_Rates;
      Filtered_Lateral_Acceleration: Missile_Accelerations;
      Rudder_Integral_Output       : Fin_Deflections;
      Fin_Command                  : Aileron_Rudder_Commands;

   begin

--    --Aileron command computations

      Filtered_Roll_Command := Roll_Command_Filter (Roll_Command);

      Roll_Error := Filtered_Roll_Command - Roll_Attitude;

      Roll_Error := Roll_Error_Limit (Roll_Error);

      Aileron_Integral_Output := Aileron_Loop.Integrate (Roll_Error);

      Filtered_Yaw_Rate := Yaw_Rate_Filter (Yaw_Rate);

      Fin_Command.Aileron_Command :=
         Aileron_Command_Limit (Aileron_Integral_Output -
                                Filtered_Yaw_Rate * Aileron_Cmd_Yaw_Rate_Gain -
                                Roll_Rate * Aileron_Cmd_Roll_Rate_Gain);

--    --Rudder command computations

      Filtered_Lateral_Acceleration :=
                                Acceleration_Filter (Lateral_Acceleration);

      Rudder_Integral_Output := Rudder_Loop.Integrate
                                   (Filtered_Lateral_Acceleration);

      Fin_Command.Rudder_Command :=
         Rudder_Command_Limit
                        (Rudder_Integral_Output +
                         (Filtered_Yaw_Rate -
                          (Gravitational_Acceleration * sin (Roll_Attitude) /
                             Missile_Velocity) -
                          Roll_Rate * Rudder_Cmd_Roll_Rate_Gain) *
                          Rudder_Cmd_Feedback_Rate_Gain
                        );

      return (Fin_Command);

   end Compute_Aileron_Rudder_Commands;
```

3.3.4.2.9.2.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Packages:

The following table summarizes the packages required by this part but defined elsewhere in the parent top level component:

Name	Type	Source	Description
Aileron_Cmd_Integral_Plus_Proportional_Gain (renamed: Aileron_Loop)	Package	Package Body	Implements integral plus proportional gain for roll command
Rudder_Cmd_Integral_Plus_Proportional_Gain (renamed: Rudder_Loop)	Package	Package Body	Implements integral plus proportional gain for acceleration feedback

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Integrate	Function	Aileron_Loop	Performs integral plus proportional gain on roll error
Integrate	Function	Rudder_Loop	Performs integral plus proportional gain on acceleration feedback

Data types:

The following table summarizes the types required by this part and defined in ancestral units:

Name	Type	Source	Description
Aileron_Rudder_Commands	Record	Package Spec.	Defines record with components for rudder and aileron commands

Data objects:

The following table summarizes the objects required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Aileron_Cmd_Yaw_Rate_Gain	Feedback_Rate_Gains	Body	Gain on yaw rate feedback in aileron command loop
Aileron_Cmd_Roll_Rate_Gain	Feedback_Rate_Gains	Body	Gain on roll rate feedback in aileron command loop
Rudder_Cmd_Roll_Rate_Gain	Rudder_Cmd_Roll_Rate_Gains	Body	Gain to roll rate feedback in rudder command loop
Rudder_Cmd_Feedback_Rate_Gain	Feedback_Rate_Gains	Body	Gain to yaw rate feedback in rudder command loop

3.3.4.2.9.2.10.2.8 LIMITATIONS

None.

3.3.4.2.9.2.10.3 UPDATE_AILERON_INTEGRATOR_GAIN UNIT DESIGN

Updates the current value of the Aileron Integrator Gain as controlled by the Aileron_Cmd_Tustin_Integrator.

3.3.4.2.9.2.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R064.

3.3.4.2.9.2.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.2.10.3.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Aileron_Integrator_Gains	Generic Float	Gains applied to roll commands

FORMAL PARAMETERS:

The following table summarizes the formal parameters required by this part:

Name	Type	Description
New_Gain	Aileron_Integrator_Gains	New value for Gain applied to roll commands

3.3.4.2.9.2.10.3.4 LOCAL DATA

None.

3.3.4.2.9.2.10.3.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.2.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Aileron_Integrator_Gain
            (New_Gain: In Roll_Command_Gains) is

begin

  Aileron_Cmd_Tustin_Integrator.Update_Gain (New_Gain => New_Gain);

end Update_Aileron_Integrator_Gain;
```

3.3.4.2.9.2.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

Packages:

The following table summarizes the packages required by this part but defined elsewhere in the parent top level component:

Name	Type	Source	Description
Aileron_Cmd_Tustin_Integrator	Package	Package Body	Implements integrator for roll command loop

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Update_ Gain	Procedure	Aileron_Cmd_Tustin_ Integrator (Pack.)	Updates value of gain in integrator

3.3.4.2.9.2.10.3.8 LIMITATIONS

None.

3.3.4.2.9.2.10.4 UPDATE_AILERON_INTEGRATOR_LIMIT UNIT DESIGN

Updates the current value of the limit on output from the aileron control loop integrator.

3.3.4.2.9.2.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement RO64.

3.3.4.2.9.2.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.2.10.4.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Fin_Deflections	Generic Float	Output from aileron control loop integrator

FORMAL PARAMETERS:

The following table summarizes the formal parameters required by this part:

Name	Type	Description
New_Limit	Fin_Deflections	New value for limit on integrator output

3.3.4.2.9.2.10.4.4 LOCAL DATA

None.

3.3.4.2.9.2.10.4.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.2.10.4.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Aileron_Integrator_Limit
            (New_Limit : in Fin_Deflections) is

begin

   Aileron_Cmd_Tustin_Integrator.Update_Limit
            (New_Absolute_Limit => New_Limit);

end Update_Aileron_Integrator_Limit;
```

3.3.4.2.9.2.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

Packages:

The following table summarizes the packages required by this part but defined elsewhere in the parent top level component:

Name	Type	Source	Description
Aileron_Cmd_Tustin_ Integrator	Package	Package Body	Implements integrator for roll command loop

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Update_ Limit	Procedure	Aileron_Cmd_Tustin_ Integrator (Pack.)	Updates value of limit in integrator

3.3.4.2.9.2.10.4.8 LIMITATIONS

None.

3.3.4.2.9.2.10.5 UPDATE_ROLL_COMMAND_PROPORTIONAL_GAIN UNIT DESIGN

Updates the current value of the Roll_Command_Proportional_Gain of of the aileron control loop

3.3.4.2.9.2.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R064.

3.3.4.2.9.2.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.2.10.5.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Roll_Gains	Generic Float	Proportional gains to roll commands

FORMAL PARAMETERS:

The following table summarizes the formal parameters required by this part:

Name	Type	Description
New_Gain	Roll_Gains	New value for proportional gain to roll commands

3.3.4.2.9.2.10.5.4 LOCAL DATA

None.

3.3.4.2.9.2.10.5.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.2.10.5.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Roll_Command_Proportional_Gain
            (New_Gain: in Roll_Command_Gains) is

begin

   Aileron_Loop.Update_Proportional_Gain (New_Proportional_Gain => New_Gain);

end Update_Roll_Command_Proportional_Gain;
```

3.3.4.2.9.2.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

Packages:

The following table summarizes the packages required by this part but defined elsewhere in the parent top level component:

Name	Type	Source	Description
Aileron_Cmd_Integral_Plus_Proportional_Gain = Aileron_Loop	Package	Package Body	Implements integral plus proportional gain for aileron loop

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Update_Proportional_Gain	Procedure	Aileron_Loop	Updates value of proportional gain

3.3.4.2.9.2.10.5.8 LIMITATIONS

None.

3.3.4.2.9.2.10.6 UPDATE_ROLL_RATE_GAIN_FOR_AILERON UNIT DESIGN

Updates the current value of the Aileron_Cmd_Roll_Rate_Gain of of the aileron control loop

3.3.4.2.9.2.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R064.

3.3.4.2.9.2.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.2.10.6.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Feedback_Rate_Gains	Generic Float	Gains to roll rate feedback for aileron commands

FORMAL PARAMETERS:

The following table summarizes the formal parameters required by this part:

Name	Type	Description
New_Gain	Feedback_Rate_Gains	New value for gain to roll rate feedback

3.3.4.2.9.2.10.6.4 LOCAL DATA

None.

3.3.4.2.9.2.10.6.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.2.10.6.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Roll_Rate_Gain_For_Aileron
             (New_Gain : in Feedback_Rate_Gains) is

begin

   Aileron_Cmd_Roll_Rate_Gain := New_Gain;

end Update_Roll_Rate_Gain_For_Aileron;
```

3.3.4.2.9.2.10.6.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.4.2.9.2.10.6.8 LIMITATIONS

None.

3.3.4.2.9.2.10.7 UPDATE_YAW_RATE_GAIN_FOR_AILERON UNIT DESIGN

Updates the current value of the Aileron_Cmd_Yaw_Rate_Gain of of the aileron control loop

3.3.4.2.9.2.10.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R064.

3.3.4.2.9.2.10.7.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.2.10.7.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Feedback_Rate_Gains	Generic Float	Gains to yaw rate feedback for aileron commands

FORMAL PARAMETERS:

The following table summarizes the formal parameters required by this part:

Name	Type	Description
New_Gain	Feedback_Rate_Gains	New value for gain to yaw rate feedback

3.3.4.2.9.2.10.7.4 LOCAL DATA

None.

3.3.4.2.9.2.10.7.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.2.10.7.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Yaw_Rate_Gain_For_Aileron
        (New_Gain : in Feedback_Rate_Gains) is

begin

   Aileron_Cmd_Yaw_Rate_Gain := New_Gain;

end Update_Yaw_Rate_Gain_For_Aileron;
```

3.3.4.2.9.2.10.7.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.4.2.9.2.10.7.8 LIMITATIONS

None.

3.3.4.2.9.2.10.8 UPDATE_RUDDER_INTEGRATOR_GAIN UNIT DESIGN

Updates the current value of the Rudder_Integrator_Gain for the integrator part of the rudder control loop integral plus propotional gain.

3.3.4.2.9.2.10.8.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R064.

3.3.4.2.9.2.10.8.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.2.10.8.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Rudder_Integrator_Gains	Generic Float	Gains applied to acceleration feedback

FORMAL PARAMETERS:

The following table summarizes the formal parameters required by this part:

Name	Type	Description
New_Gain	Rudder_integrator_Gains	New value for Gain applied to acceleration

3.3.4.2.9.2.10.8.4 LOCAL DATA

None.

3.3.4.2.9.2.10.8.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.2.10.8.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Rudder_Integrator_Gain
        (New_Gain : in Acceleration_Gains) is

begin

   Rudder_Cmd_Tustin_Integrator.Update_Gain (New_Gain => New_Gain);

end Update_Rudder_Integrator_Gain;
```

3.3.4.2.9.2.10.8.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

Packages:

The following table summarizes the packages required by this part but defined elsewhere in the parent top level component:

Name	Type	Source	Description
Rudder_Cmd_Tustin_ Integrator	Package	Package Body	Implements tustin integrator for rudder loop

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Update_Gain	Procedure	Rudder_Cmd_Tustin_ Integrator	Updates value of integral gain

3.3.4.2.9.2.10.8.8 LIMITATIONS

None.

3.3.4.2.9.2.10.9 UPDATE_RUDDER_INTEGRATOR_LIMIT UNIT DESIGN

Updates the current value of the limit on output from the rudder control loop integrator.

3.3.4.2.9.2.10.9.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R064.

3.3.4.2.9.2.10.9.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.2.10.9.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Fin_Deflections	Generic Float	Output from rudder control loop integrator

FORMAL PARAMETERS:

The following table summarizes the formal parameters required by this part:

Name	Type	Description
New_Limit	Fin_Deflections	New value for limit on integrator output

3.3.4.2.9.2.10.9.4 LOCAL DATA

None.

3.3.4.2.9.2.10.9.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.2.10.9.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Rudder_Integrator_Limit
        (New_Limit : in Fin_Deflections) is

begin
```

```
      Rudder_Cmd_Tustin_Integrator.Update_Limit
                                    (New_Absolute_Limit => New_Limit);

   end Update_Rudder_Integrator_Limit;
```

3.3.4.2.9.2.10.9.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

Packages:

The following table summarizes the packages required by this part but defined elsewhere in the parent top level component:

Name	Type	Source	Description
Rudder_Cmd_Tustin_ Integrator	Package	Package Body	Implements tustin int- egrator for rudder loop

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Update_ Limit	Procedure	Rudder_Cmd_Tustin_ Integrator	Updates value of limit on integrator output

3.3.4.2.9.2.10.9.8 LIMITATIONS

None.

3.3.4.2.9.2.10.10 UPDATE_FEEDBACK_RATE_GAIN_FOR_RUDDER UNIT DESIGN

Updates the current value of the Rudder_Cmd_Feedback_Rate_Gain of the yaw rate for the rudder control loop

3.3.4.2.9.2.10.10.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R064.

3.3.4.2.9.2.10.10.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.2.10.10.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Feedback_Rate_Gains	Generic Float	Gains to yaw rate feedback for rudder commands

FORMAL PARAMETERS:

The following table summarizes the formal parameters required by this part:

Name	Type	Description
New_Gain	Feedback_Rate_Gains	New value for gain to yaw rate feedback

3.3.4.2.9.2.10.10.4 LOCAL DATA

None.

3.3.4.2.9.2.10.10.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.2.10.10.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Feedback_Rate_Gain_For_Rudder
            (New_Gain : In Feedback_Rate_Gains) is

begin

   Rudder_Cmd_Feedback_Rate_Gain := New_Gain;

end Update_Feedback_Rate_Gain_For_Rudder;
```

3.3.4.2.9.2.10.10.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.4.2.9.2.10.10.8 LIMITATIONS

None.

3.3.4.2.9.2.10.11 UPDATE_ROLL_RATE_GAIN_FOR_RUDDER UNIT DESIGN

Updates the current value of the Rudder_Cmd_Roll_Rate_Gain for the rudder control loop

3.3.4.2.9.2.10.11.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R064.

3.3.4.2.9.2.10.11.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.2.10.11.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Rudder_Cmd_Roll_Rate_ Gains	Generic Float	Gains to roll rate feedback for rudder commands

FORMAL PARAMETERS:

The following table summarizes the formal parameters required by this part:

Name	Type	Description
New_Gain	Feedback_Rate_ Gains	New value for gain to roll rate feedback

3.3.4.2.9.2.10.11.4 LOCAL DATA

None.

3.3.4.2.9.2.10.11.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.2.10.11.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Roll_Rate_Gain_For_Rudder
            (New_Gain : in Rudder_Cmd_Roll_Rate_Gains) is

begin

   Rudder_Cmd_Roll_Rate_Gain := New_Gain;

end Update_Roll_Rate_Gain_For_Rudder;
```

3.3.4.2.9.2.10.11.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.4.2.9.2.10.11.8 LIMITATIONS

None.

3.3.4.2.9.2.10.12 UPDATE_ACCELERATION_PROPORTIONAL_GAIN UNIT DESIGN

Updates the current value of the Acceleration_Proportional_Gain of of the rudder control loop

3.3.4.2.9.2.10.12.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R064.

3.3.4.2.9.2.10.12.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.2.10.12.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Acceleration_Gains	Generic Float	Proportional gains to acceleration feedback

FORMAL PARAMETERS:

The following table summarizes the formal parameters required by this part:

Name	Type	Description
New_Gain	Roll_Gains	New value for proportional gain to acceleration

3.3.4.2.9.2.10.12.4 LOCAL DATA

None.

3.3.4.2.9.2.10.12.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.2.10.12.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Acceleration_Proportional_Gain
            (New_Gain : in Acceleration_Gains) is

begin

   Rudder_Loop.Update_Proportional_Gain (New_Proportional_Gain => New_Gain);

end Update_Acceleration_Proportional_Gain;
```

3.3.4.2.9.2.10.12.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

Packages:

The following table summarizes the packages required by this part but defined elsewhere in the parent top level component:

Name	Type	Source	Description
Rudder_Cmd_Integral_Plus_Proportional_Gain = Rudder_Loop	Package	Package Body	Implements integral plus proportional gain for rudder loop

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Update_Proportional_Gain	Procedure	Rudder_Loop	Updates value of proportional gain

3.3.4.2.9.2.10.12.8 LIMITATIONS

None.

3.3.4.2.9.3 PITCH AUTOPILOT PACKAGE DESIGN (CATALOG #P307-0)

This package body implements the Pitch Autopilot function. It contains the instantiation of the Integral Plus Proportional Gain package for the integrator loop of the Normal Acceleration error, as well as subprogram bodies for operations declared in the package specification.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.4.2.9.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R059.

3.3.4.2.9.3.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.3.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Normal_Acceleration_Commands	Generic Float	Type for input commands
Acceleration_Command_Gains	Generic Float	Gains used in Integral_Plus_Proportional_Gain
Acceleration_Gains	Generic Float	Gains applied to filtered acceleration feedback
Fin_Deflections	Generic Float	Type for Fin Deflection command
Pitch_Rate_Gains	Generic Float	Gains applied to filtered pitch rate

Data objects:

The following table summarizes the generic formal objects required by this part:

Initial_Integrator_Gain	Acceleration_Command_Gains	Initial gain to Tustin integrator input
Initial_Integrator_Limit	Fin_Deflections	Initial limit on Tustin integrator output
Initial_Acceleration_Gain	Acceleration_Gains	Initial gain to filtered acceleration feedback
Initial_Pitch_Rate_Gain	Pitch_Rate_Gains	Initial gain to filtered pitch rate feedback
Initial_Proportional_Gain	Acceleration_Command_Gains	Initial proportional gain for integral loop

3.3.4.2.9.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Pitch_Rate_Gain	Pitch_Rate_Gains	Pitch rate feedback gain
Acceleration_Gain	Acceleration_Gains	Accel. feedback gain

3.3.4.2.9.3.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.3.6 PROCESSING

The following describes the processing performed by this part:

```
with Signal_Processing;
separate (Autopilot)
package body Pitch_Autopilot is

   Acceleration_Gain : Acceleration_Gains := Initial_Acceleration_Gain;

   Pitch_Rate_Gain   : Pitch_Rate_Gains   := Initial_Pitch_Rate_Gain;

   package Tustin_Integrator is new
             Signal_Processing.Tustin_Integrator_With_Limit
                (Signals               => Normal_Acceleration_Commands,
                 States                => Fin_Deflections,
                 Gained_Signals        => Fin_Deflections,
                 Gains                 => Acceleration_Command_Gains,
                 Initial_Tustin_Gain   => Initial_Integrator_Gain,
                 Initial_Signal_Level  => 0.0,
                 Initial_Signal_Limit  => Initial_Integrator_Limit);

   package Pitch_Loop_Integral_Plus_Proportional_Gain is new
      Integral_Plus_Proportional_Gain
         (Input_Signals             => Normal_Acceleration_Commands,
          Gains                     => Acceleration_Command_Gains,
          Integrated_Signals        => Fin_Deflections,
          Initial_Proportional_Gain => Initial_Proportional_Gain,
          Tustin_Integrate          => Tustin_Integrator.Integrate);

   package Pitch_Loop renames Pitch_Loop_Integral_Plus_Proportional_Gain;

end Pitch_Autopilot;
```

3.3.4.2.9.3.7 UTILIZATION OF OTHER ELEMENTS

The following library units are with'd by this part:
    1. Signal Processing

UTILIZATION OF EXTERNAL ELEMENTS:

Packages:

The following table summarizes the external packages required by this part:

Name	Type	Source	Description
Tustin_Integrator_With_Limit	Generic Package	1.	Exports integrate function for Integral_Plus_Proportional_Gain instantiation

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Packages:

The following table summarizes the packages required by this part but defined elsewhere in the parent top level component:

Name	Type	Source	Description
Integral_Plus_Proportional_Gain	Generic Package	Package Spec.	Performs integrator function on normal acceleration error

3.3.4.2.9.3.8 LIMITATIONS

None.

3.3.4.2.9.3.9 LLCSC DESIGN

None.

3.3.4.2.9.3.10 UNIT DESIGN

3.3.4.2.9.3.10.1 INITIALIZE_PITCH_AUTOPILOT UNIT DESIGN

Initializes state of integrator in pitch control loop.

3.3.4.2.9.3.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R053 (2).

3.3.4.2.9.3.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.3.10.1.3 INPUT/OUTPUT

None.

3.3.4.2.9.3.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Gained_Signal	Fin_Deflections	Value of acceleration command after applying gain
Intitial_State	Fin_Deflections	Output state of integrator loop

3.3.4.2.9.3.10.1.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.3.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   procedure Initialize_Pitch_Autopilot
               (Normal_Acceleration_Command : in Normal_Acceleration_Commands;
               Measured_Normal_Acceleration : in Accelerations;
               Measured_Pitch_Rate          : in Pitch_Rates;
               Initial_Elevator_Command     : in Fin_Deflections) is

      Gained_Signal : Fin_Deflections;

      Initial_State : Fin_Deflections;

   begin

      Gained_Signal := (- Normal_Acceleration_Command +
                        Measured_Normal_Acceleration) *
                        Initial_Proportional_Gain;

      Initial_State :=
                  Gained_Signal +
                  Measured_Normal_Acceleration * Acceleration_Gain -
                  Measured_Pitch_Rate * Pitch_Rate_Gain +
                  Initial_Elevator_Command;

      Tustin_Integrator.Reset (Integrator_State => Initial_State,
                               Signal           =>
                                 Normal_Acceleration_Commands (0.0));

    end Initialize_Pitch_Autopilot;
```

3.3.4.2.9.3.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Reset	Procedure	Tustin_Integrator from package body	Resets state of integrator
"+"	Function	Generic Package Spec	Adds acceleration cmd and measured acceleration feedback
"*"	Function	Generic Package Spec	Times for applying gain to acceleration command
"*"	Function	Generic Package Spec	Times for applying gain to acceleration feedback
"*"	Function	Generic Package Spec	Times for applying gain to pitch rate feedback

Data types:

The following table summarizes the types required by this part and defined in ancestral units:

Name	Type	Source	Description
Normal_Acceleration_Commands	Generic Float	Generic Pkg Spec	Type for input acceleration command
Accelerations	Generic Float	Generic Pkg Spec	Type for acceleration feedbacks
Pitch_Rates	Generic Float	Generic Pkg Spec	Type for pitch rate feedback
Fin_Deflections	Generic Float	Generic Pkg Spec	Data type for state of integrator

Data objects:

The following table summarizes the objects required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Initial_Integrator_Gain	Acceleration_Command_Gains	Package Body	Gain applied to acceleration error
Acceleration_Gain	Acceleration_Gains	Package Body	Gain applied to acceleration feedback
Pitch_Rate_Gain	Pitch_Rate_Gains	Package Body	Gain applied to pitch rate feedback

3.3.4.2.9.3.10.1.8 LIMITATIONS

None.

3.3.4.2.9.3.10.2 COMPUTE_ELEVATOR_COMMAND UNIT DESIGN

Computes elevator fin deflection command.

3.3.4.2.9.3.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R053 (2).

3.3.4.2.9.3.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.3.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table summarizes the formal parameters required by this part:

Name	Type	Description
Normal_Acceleration_Command	Normal_Acceleration_Commands	Input command from guidance system
Measured_Normal_Accelerations	Accelerations	Measured acceleration feedback
Measured_Pitch_Rate	Pitch_Rates	Measured pitch rate feedback
<return value>	Fin Deflections	Elevator Command

3.3.4.2.9.3.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Filtered_Normal_Acceleration	Accelerations	Value of acceleration feedback after applying filter
Normal_Acceleration_Error	Normal_Acceleration_Commands	Difference between input acceleration command and filtered acceleration feedback
Integral_Output	Fin_Deflections	Output state of integrator loop
Filtered_Pitch_Rate	Pitch_Rates	Value of pitch rate feedback after applying filter
Limited_Elevator_Command	Fin_Deflections	Output value from Compute_Elevator_Command

3.3.4.2.9.3.10.2.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.3.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
function Compute_Elevator_Command
            (Normal_Acceleration_Command  : in Normal_Acceleration_Commands;
             Measured_Normal_Acceleration : in Accelerations;
             Measured_Pitch_Rate          : in Pitch_Rates)
   return Fin_Deflections is

   Filtered_Normal_Acceleration : Accelerations;
   Normal_Acceleration_Error    : Normal_Acceleration_Commands;
   Integral_Output              : Fin_Deflections;
   Filtered_Pitch_Rate          : Pitch_Rates;
   Limited_Elevator_Command     : Fin_Deflections;

begin

   Filtered_Normal_Acceleration := Acceleration_Filter (Measured_Normal_Acceleration

   Normal_Acceleration_Error :=  Normal_Acceleration_Command -
                                 Filtered_Normal_Acceleration;

   Integral_Output := Pitch_Loop.Integrate (Normal_Acceleration_Error);

   Filtered_Pitch_Rate := Pitch_Rate_Filter (Measured_Pitch_Rate);

   Limited_Elevator_Command :=
               Limit (Integral_Output -
                      Filtered_Normal_Acceleration  * Acceleration_Gain +
                      Filtered_Pitch_Rate * Pitch_Rate_Gain);

   return (Limited_Elevator_Command);
```

```
   end Compute_Elevator_Command;
```

3.3.4.2.9.3.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Packages:

The following table summarizes the packages required by this part but defined elsewhere in the parent top level component:

Name	Type	Source	Description
Pitch_Loop_Integral_Plus_Proportional_Gain (renamed: Pitch_Loop)	Package	Package Body	Implements integrator loop fcr normal acceleration error

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Acceleration_Filter	Function	Generic Pkg Spec	Performs filter function on Acceleration feedback
"-"	Function	Generic Pkg Spec	Minus for calculating normal acceleration error in integral loop
Integrate	Function	Pitch Loop package in body	Performs integral plus proportional gain function
Pitch_Rate Filter	Function	Generic Pkg Spec	Performs filter function on pitch rate feedback
Limit	Function	Generic Pkg Spec	Performs Limiter function (e.g. R202)
"*"	Function	Generic Pkg Spec	Times for applying gain to acceleration feedback
"*"	Function	Generic Pkg Spec	Times for applying gain to Pitch Rate feedback

Data types:

The following table summarizes the data types required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Normal_Acceleration_Commands	Generic Float	Generic Pkg Spec	Type for input commands
Accelerations	Generic Float	Generic Pkg Spec	Type for acceleration feedbacks
Pitch_Rates	Generic Float	Generic Pkg Spec	Type for pitch rate feedback
Fin_Deflections	Generic Float	Generic Pkg Spec	Type for Fin Deflection output

Data objects:

The following table summarizes the objects required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Acceleration_Gain	Acceleration_Gains	Package Body	Gain applied to acceleration feedback
Pitch_Rate_Gain	Pitch_Rate_Gains	Package Body	Gain applied to pitch rate feedback

3.3.4.2.9.3.10.2.8 LIMITATIONS

None.

3.3.4.2.9.3.10.3 UPDATE_PITCH_RATE_GAIN UNIT DESIGN

Updates the current value of the Pitch_Rate_Gain

3.3.4.2.9.3.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R053.

3.3.4.2.9.3.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.3.10.3.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Pitch_Rate_Gains	Generic Float	Gains applied to filtered pitch rate

FORMAL PARAMETERS:

The following table summarizes the formal parameters required by this part:

Name	Type	Description
New_Gain	Pitch_Rate_Gains	New value for Gain applied to filtered pitch rate

3.3.4.2.9.3.10.3.4 LOCAL DATA

None.

3.3.4.2.9.3.10.3.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.3.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Pitch_Rate_Gain (New_Gain: in Pitch_Rate_Gains) is

begin

   Pitch_Rate_Gain := New_Gain;

end Update_Pitch_Rate_Gain;
```

3.3.4.2.9.3.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data objects:

The following table summarizes the objects required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Pitch_Rate_Gain	Pitch_Rate_Gains	Package Body	Pitch rate feedback gain

3.3.4.2.9.3.10.3.8 LIMITATIONS

None.

3.3.4.2.9.3.10.4 UPDATE_ACCELERATION_GAIN UNIT DESIGN

Updates the current value of the Acceleration_Gain

3.3.4.2.9.3.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R053.

3.3.4.2.9.3.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.3.10.4.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Acceleration_Gains	Generic Float	Gains applied to filtered acceleration feedback

FORMAL PARAMETERS:

The following table summarizes the formal parameters required by this part:

Name	Type	Description
New_Gain	Acceleration_Gains	New value for Gain applied to filtered acceleration feedback

3.3.4.2.9.3.10.4.4 LOCAL DATA

None.

3.3.4.2.9.3.10.4.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.3.10.4.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Acceleration_Gain (New_Gain: in Acceleration_Gains) is
begin
   Acceleration_Gain := New_Gain;
end Update_Acceleration_Gain;
```

3.3.4.2.9.3.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data objects:

The following table summarizes the objects required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Acceleration_ Gain	Acceleration_ Gains	Package Body	Acceleration feedback gain

3.3.4.2.9.3.10.4.8 LIMITATIONS

None.

3.3.4.2.9.3.10.5 UPDATE_INTEGRATOR_GAIN UNIT DESIGN

Updates the current value of the Integrator_Gain

3.3.4.2.9.3.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R053.

3.3.4.2.9.3.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.3.10.5.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Acceleration_Command_Gains	Generic Float	Gains applied to acceleration commands

FORMAL PARAMETERS:

The following table summarizes the formal parameters required by this part:

Name	Type	Description
New_Gain	Acceleration_Command_Gains	New value for Gain applied to acceleration commands

3.3.4.2.9.3.10.5.4 LOCAL DATA

None.

3.3.4.2.9.3.10.5.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.3.10.5.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Integrator_Gain (New_Gain: in Acceleration_Command_Gains) is
begin
   Tustin_Integrator.Update_Gain (New_Gain => New_Gain);
```

```
end Update_Integrator_Gain;
```

3.3.4.2.9.3.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Packages:

The following table summarizes the packages required by this part but defined elsewhere in the parent top level component:

Name	Type	Source	Description
Tustin_Integrator	Package	Package Body	Implements integrator function in pitch loop function

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Update_Gain	Procedure	Pitch_Loop package in body	Updates integrator gain in pitch loop function

3.3.4.2.9.3.10.5.8 LIMITATIONS

None.

3.3.4.2.9.3.10.6 UPDATE_INTEGRATOR_LIMIT UNIT DESIGN

Updates the current value of the Integrator_Limit

3.3.4.2.9.3.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R053.

3.3.4.2.9.3.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.3.10.6.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Fin_Deflections	Generic Float	Value for fin deflection output from package

FORMAL PARAMETERS:

The following table summarizes the formal parameters required by this part:

Name	Type	Description
New_Limit	Fin_Deflections	New value for limit on fin deflection output

3.3.4.2.9.3.10.6.4 LOCAL DATA

None.

3.3.4.2.9.3.10.6.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.3.10.6.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Integrator_Limit (New_Limit: in Fin_Deflections) is
begin
   Tustin_Integrator.Update_Limit (New_Absolute_Limit => New_Limit);
end Update_Integrator_Limit;
```

3.3.4.2.9.3.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Packages:

The following table summarizes the packages required by this part but defined elsewhere in the parent top level component:

Name	Type	Source	Description
Tustin_Integrator	Package	Package Body	Implements integrator function in pitch loop function

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Update_Limit	Procedure	Pitch_Loop package in body	Updates integrator limit in pitch loop function

3.3.4.2.9.3.10.6.8 LIMITATIONS

None.

3.3.4.2.9.3.10.7 UPDATE_PROPORTIONAL_GAIN UNIT DESIGN

Updates the current value of the Proporational_Gain

3.3.4.2.9.3.10.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement RO53.

3.3.4.2.9.3.10.7.2 LOCAL ENTITIES DESIGN

None.

3.3.4.2.9.3.10.7.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table summarizes the generic formal types required by this part:

Name	Type	Description
Acceleration_Command_Gains	Generic Float	Gains applied to normal acceleration commands

FORMAL PARAMETERS:

The following table summarizes the formal parameters required by this part:

Name	Type	Description
New_Proportional_Gain	Acceleration_Command_Gains	New value for Gain applied to Normal_Acceleration_Commands

3.3.4.2.9.3.10.7.4 LOCAL DATA

None.

3.3.4.2.9.3.10.7.5 PROCESS CONTROL

Not applicable.

3.3.4.2.9.3.10.7.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Proportional_Gain
        (New_Proportional_Gain : in Acceleration_Command_Gains) is

begin

   Pitch_Loop.Update_Proportional_Gain
                          (New_Proportional_Gain => New_Proportional_Gain);

end Update_Proportional_Gain;
```

3.3.4.2.9.3.10.7.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Packages:

The following table summarizes the packages required by this part but defined elsewhere in the parent top level component:

Name	Type	Source	Description
Pitch_Loop_Integral_Plus_Proportional_Gain = Pitch_Loop	Package	Package Body	Implements integral plus proportional gain in pitch loop

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Update_Proportional_Gain	Procedure	Pitch_Loop package in body	Updates proportional gain in pitch loop function

3.3.4.2.9.3.10.7.8 LIMITATIONS

None.

3.3.4.2.10 UNIT DESIGN

None.

```
package body Autopilot is

   package body Integral_Plus_Proportional_Gain is separate;

   package body Pitch_Autopilot is separate;

   package body Lateral_Directional_Autopilot is separate;

end Autopilot;
```

```
separate (Autopilot)
package body Integral_Plus_Proportional_Gain is

   Proportional_Gain: Gains := Initial_Proportional_Gain;


pragma PAGE;
   function Integrate (Signal: Input_Signals) return Integrated_Signals is

   begin

      return (Tustin_Integrate (Signal) + Signal * Proportional_Gain);

   end Integrate;

pragma PAGE;
   procedure Update_Proportional_Gain (New_Proportional_Gain : in Gains) is

   begin

     Proportional_Gain := New_Proportional_Gain;

   end Update_Proportional_Gain;

end Integral_Plus_Proportional_Gain;
```

```
with Signal_Processing;
separate (Autopilot)
package body Lateral_Directional_Autopilot is

-- --Initial vaules for Aileron Control Loop

   Aileron_Cmd_Roll_Rate_Gain     : Feedback_Rate_Gains :=
                                       Initial_Roll_Rate_Gain_For_Aileron;
   Aileron_Cmd_Yaw_Rate_Gain      : Feedback_Rate_Gains :=
                                       Initial_Yaw_Rate_Gain_For_Aileron;

-- --Initial values for rudder control loop

   Rudder_Cmd_Roll_Rate_Gain      : Rudder_Cmd_Roll_Rate_Gains :=
                                       Initial_Roll_Rate_Gain_For_Rudder;
   Rudder_Cmd_Feedback_Rate_Gain  : Feedback_Rate_Gains :=
                                       Initial_Yaw_Rate_Gain_For_Rudder;

-- Packages for Aeliron control loop

   package Aileron_Cmd_Tustin_Integrator is new
             Signal_Processing.Tustin_Integrator_With_Limit
                (Signals              => Roll_Commands,
                 States               => Fin_Deflections,
                 Gained_Signals       => Fin_Deflections,
                 Gains                => Roll_Command_Gains,
                 Initial_Tustin_Gain  => Initial_Aileron_Integrator_Gain,
                 Initial_Signal_Level => 0.0,
                 Initial_Signal_Limit => Initial_Aileron_Integrator_Limit);

   package Aileron_Cmd_Integral_Plus_Proportional_Gain is new
      Integral_Plus_Proportional_Gain
         (Input_Signals             => Roll_Commands,
          Gains                     => Roll_Command_Gains,
          Integrated_Signals        => Fin_Deflections,
          Initial_Proportional_Gain => Initial_Roll_Command_Proportional_Gain,
          Tustin_Integrate          => Aileron_Cmd_Tustin_Integrator.Integrate);

   package Aileron_Loop renames Aileron_Cmd_Integral_Plus_Proportional_Gain;

-- Packages for rudder control loop

   package Rudder_Cmd_Tustin_Integrator is new
             Signal_Processing.Tustin_Integrator_With_Limit
                (Signals              => Missile_Accelerations,
                 States               => Fin_Deflections,
                 Gained_Signals       => Fin_Deflections,
                 Gains                => Acceleration_Gains,
                 Initial_Tustin_Gain  => Initial_Rudder_Integrator_Gain,
                 Initial_Signal_Level => 0.0,
                 Initial_Signal_Limit => Initial_Rudder_Integrator_Limit);

   package Rudder_Cmd_Integral_Plus_Proportional_Gain is new
      Integral_Plus_Proportional_Gain
         (Input_Signals             => Missile_Accelerations,
          Gains                     => Acceleration_Gains,
          Integrated_Signals        => Fin_Deflections,
```

```
        Initial_Proportional_Gain => Initial_Acceleration_Proportional_Gain,
        Tustin_Integrate          => Rudder_Cmd_Tustin_Integrator.Integrate);

   package Rudder_Loop renames Rudder_Cmd_Integral_Plus_Proportional_Gain;

                                                                  pragma PAGE;
   procedure Initialize_Lateral_Directional_Autopilot
               (Initial_Aileron_Command    : in Fin_Deflections;
                Initial_Rudder_Command     : in Fin_Deflections;
                Gravitational_Acceleration : in Gravitational_Accelerations;
                Roll_Command               : in Roll_Commands;
                Roll_Attitude              : in Roll_Attitudes;
                Roll_Rate                  : in Feedback_Rates;
                Yaw_Rate                   : in Feedback_Rates;
                Missile_Velocity           : in Velocities;
                Lateral_Acceleration       : in Missile_Accelerations) is

      Gained_Roll_Command_Signal : Fin_Deflections;

      Initial_Aileron_State : Fin_Deflections;

      Initial_Rudder_State : Fin_Deflections;

   begin

      Gained_Roll_Command_Signal :=
              (Roll_Command - Roll_Attitude) *
              Initial_Roll_Command_Proportional_Gain;

      Initial_Aileron_State :=
                Initial_Aileron_Command -
                Gained_Roll_Command_Signal +
                Yaw_Rate * Aileron_Cmd_Yaw_Rate_Gain +
                Roll_Rate * Aileron_Cmd_Yaw_Rate_Gain;

      Initial_Rudder_State :=
                Initial_Rudder_Command -
                (Yaw_Rate -
                 (Gravitational_Acceleration * Sin (Roll_Attitude)
                          / Missile_Velocity) -
                 (Roll_Rate * Rudder_Cmd_Roll_Rate_Gain)
                ) * Rudder_Cmd_Feedback_Rate_Gain -
                Lateral_Acceleration * Initial_Acceleration_Proportional_Gain;

      Aileron_Cmd_Tustin_Integrator.RESET
         (Integrator_State => Initial_Aileron_State,
          Signal           => 0.0);

      Rudder_Cmd_Tustin_Integrator.RESET
         (Integrator_State => Initial_Rudder_State,
          Signal           => 0.0);

    end Initialize_Lateral_Directional_Autopilot;


                                                                  pragma PAGE;
   function Compute_Aileron_Rudder_Commands
```

```
            (Roll_Command                : in Roll_Commands;
             Roll_Attitude               : in Roll_Attitudes;
             Roll_Rate                   : in Feedback_Rates;
             Yaw_Rate                    : in Feedback_Rates;
             Lateral_Acceleration        : in Missile_Accelerations;
             Missile_Velocity            : in Velocities;
             Gravitational_Acceleration : in Gravitational_Accelerations)
         return Aileron_Rudder_Commands is

      Filtered_Roll_Command        : Roll_Commands;
      Roll_Error                   : Roll_Commands;
      Aileron_Integral_Output      : Fin_Deflections;
      Filtered_Yaw_Rate            : Feedback_Rates;
      Filtered_Lateral_Acceleration: Missile_Accelerations;
      Rudder_Integral_Output       : Fin_Deflections;
      Fin_Command                  : Aileron_Rudder_Commands;

   begin

--    --Aileron command computations

      Filtered_Roll_Command := Roll_Command_Filter (Roll_Command);

      Roll_Error := Filtered_Roll_Command - Roll_Attitude;

      Roll_Error := Roll_Error_Limit (Roll_Error);

      Aileron_Integral_Output := Aileron_Loop.Integrate (Roll_Error);

      Filtered_Yaw_Rate := Yaw_Rate_Filter (Yaw_Rate);

      Fin_Command.Aileron_Command :=
         Aileron_Command_Limit (Aileron_Integral_Output -
                                Filtered_Yaw_Rate * Aileron_Cmd_Yaw_Rate_Gain -
                                Roll_Rate * Aileron_Cmd_Roll_Rate_Gain);

--    --Rudder command computations

      Filtered_Lateral_Acceleration :=
                                Acceleration_Filter (Lateral_Acceleration);

      Rudder_Integral_Output := Rudder_Loop.Integrate
                                   (Filtered_Lateral_Acceleration);

      Fin_Command.Rudder_Command :=
         Rudder_Command_Limit
                        (Rudder_Integral_Output +
                         (Filtered_Yaw_Rate -
                          (Gravitational_Acceleration * Sin (Roll_Attitude) /
                             Missile_Velocity) -
                          Roll_Rate * Rudder_Cmd_Roll_Rate_Gain) *
                          Rudder_Cmd_Feedback_Rate_Gain
                        );

      return (Fin_Command);

   end Compute_Aileron_Rudder_Commands;
```

```
                                                                    pragma PAGE;
procedure Update_Aileron_Integrator_Gain
            (New_Gain: in Roll_Command_Gains) is

begin

  Aileron_Cmd_Tustin_Integrator.Update_Gain (New_Gain => New_Gain);

end Update_Aileron_Integrator_Gain;

                                                                    pragma PAGE;
procedure Update_Aileron_Integrator_Limit
            (New_Limit : in Fin_Deflections) is

begin

   Aileron_Cmd_Tustin_Integrator.Update_Limit
            (New_Absolute_Limit => New_Limit);

end Update_Aileron_Integrator_Limit;
                                                                    pragma PAGE;
procedure Update_Roll_Command_Proportional_Gain
            (New_Gain: in Roll_Command_Gains) is

begin

   Aileron_Loop.Update_Proportional_Gain (New_Proportional_Gain => New_Gain);

end Update_Roll_Command_Proportional_Gain;
                                                                    pragma PAGE;
procedure Update_Roll_Rate_Gain_For_Aileron
            (New_Gain : in Feedback_Rate_Gains) is

begin

   Aileron_Cmd_Roll_Rate_Gain := New_Gain;

end Update_Roll_Rate_Gain_For_Aileron;
                                                                    pragma PAGE;
procedure Update_Yaw_Rate_Gain_For_Aileron
        (New_Gain : in Feedback_Rate_Gains) is

begin

   Aileron_Cmd_Yaw_Rate_Gain := New_Gain;

end Update_Yaw_Rate_Gain_For_Aileron;

                                                                    pragma PAGE;
procedure Update_Rudder_Integrator_Gain
        (New_Gain : in Acceleration_Gains) is

begin

   Rudder_Cmd_Tustin_Integrator.Update_Gain (New_Gain => New_Gain);
```

```
   end Update_Rudder_Integrator_Gain;

                                                                    pragma PAGE;
   procedure Update_Rudder_Integrator_Limit
            (New_Limit : in Fin_Deflections) is

   begin

      Rudder_Cmd_Tustin_Integrator.Update_Limit
                                    (New_Absolute_Limit => New_Limit);

   end Update_Rudder_Integrator_Limit;

                                                                    pragma PAGE;
   procedure Update_Feedback_Rate_Gain_For_Rudder
               (New_Gain : in Feedback_Rate_Gains) is

   begin

      Rudder_Cmd_Feedback_Rate_Gain := New_Gain;

   end Update_Feedback_Rate_Gain_For_Rudder;


                                                                    pragma PAGE;
   procedure Update_Roll_Rate_Gain_For_Rudder
               (New_Gain : in Rudder_Cmd_Roll_Rate_Gains) is

   begin

      Rudder_Cmd_Roll_Rate_Gain := New_Gain;

   end Update_Roll_Rate_Gain_For_Rudder;

                                                                    pragma PAGE;
   procedure Update_Acceleration_Proportional_Gain
               (New_Gain : in Acceleration_Gains) is

   begin

      Rudder_Loop.Update_Proportional_Gain (New_Proportional_Gain => New_Gain);

   end Update_Acceleration_Proportional_Gain;

end Lateral_Directional_Autopilot;
```

```
with Signal_Processing;
separate (Autopilot)
package body Pitch_Autopilot is

   Acceleration_Gain : Acceleration_Gains := Initial_Acceleration_Gain;

   Pitch_Rate_Gain   : Pitch_Rate_Gains   := Initial_Pitch_Rate_Gain;

   package Tustin_Integrator is new
             Signal_Processing.Tustin_Integrator_With_Limit
                (Signals               => Normal_Acceleration_Commands,
                 States                => Fin_Deflections,
                 Gained_Signals        => Fin_Deflections,
                 Gains                 => Acceleration_Command_Gains,
                 Initial_Tustin_Gain   => Initial_Integrator_Gain,
                 Initial_Signal_Level  => 0.0,
                 Initial_Signal_Limit  => Initial_Integrator_Limit);

   package Pitch_Loop_Integral_Plus_Proportional_Gain is new
      Integral_Plus_Proportional_Gain
         (Input_Signals             => Normal_Acceleration_Commands,
          Gains                     => Acceleration_Command_Gains,
          Integrated_Signals        => Fin_Deflections,
          Initial_Proportional_Gain => Initial_Proportional_Gain,
          Tustin_Integrate          => Tustin_Integrator.Integrate);

   package Pitch_Loop renames Pitch_Loop_Integral_Plus_Proportional_Gain;

                                                                  pragma PAGE;
   procedure Initialize_Pitch_Autopilot
               (Normal_Acceleration_Command : in Normal_Acceleration_Commands;
               Measured_Normal_Acceleration : in Accelerations;
               Measured_Pitch_Rate          : in Pitch_Rates;
               Initial_Elevator_Command     : in Fin_Deflections) is

      Gained_Signal : Fin_Deflections;

      Initial_State : Fin_Deflections;

   begin

      Gained_Signal := (- Normal_Acceleration_Command +
                        Measured_Normal_Acceleration) *
                        Initial_Proportional_Gain;

      Initial_State :=
                 Gained_Signal +
                 Measured_Normal_Acceleration * Acceleration_Gain -
                 Measured_Pitch_Rate * Pitch_Rate_Gain +
                 Initial_Elevator_Command;

      Tustin_Integrator.RESET (Integrator_State => Initial_State,
                               Signal           =>
                                 Normal_Acceleration_Commands (0.0));

    end Initialize_Pitch_Autopilot;
```

```
                                                                    pragma PAGE;
function Compute_Elevator_Command
            (Normal_Acceleration_Command  : in Normal_Acceleration_Commands;
             Measured_Normal_Acceleration : in Accelerations;
             Measured_Pitch_Rate          : in Pitch_Rates)
   return Fin_Deflections is

   Filtered_Normal_Acceleration : Accelerations;
   Normal_Acceleration_Error    : Normal_Acceleration_Commands;
   Integral_Output              : Fin_Deflections;
   Filtered_Pitch_Rate          : Pitch_Rates;
   Limited_Elevator_Command     : Fin_Deflections;

begin

   Filtered_Normal_Acceleration := Acceleration_Filter (Measured_Normal_Acceleration

   Normal_Acceleration_Error :=  Normal_Acceleration_Command -
                                 Filtered_Normal_Acceleration;

   Integral_Output := Pitch_Loop.Integrate (Normal_Acceleration_Error);

   Filtered_Pitch_Rate := Pitch_Rate_Filter (Measured_Pitch_Rate);

   Limited_Elevator_Command :=
               Limit (Integral_Output -
                      Filtered_Normal_Acceleration  * Acceleration_Gain +
                      Filtered_Pitch_Rate * Pitch_Rate_Gain);

   return (Limited_Elevator_Command);

end Compute_Elevator_Command;

                                                                  pragma PAGE;
procedure Update_Pitch_Rate_Gain (New_Gain: in Pitch_Rate_Gains) is

begin

   Pitch_Rate_Gain := New_Gain;

end Update_Pitch_Rate_Gain;

                                                                  pragma PAGE;
procedure Update_Acceleration_Gain (New_Gain: in Acceleration_Gains) is

begin

   Acceleration_Gain := New_Gain;

end Update_Acceleration_Gain;
                                                                  pragma PAGE;
procedure Update_Integrator_Gain (New_Gain: in Acceleration_Command_Gains) is

begin

   Tustin_Integrator.Update_Gain (New_Gain => New_Gain);
```

```
   end Update_Integrator_Gain;
                                                                    pragma PAGE;
   procedure Update_Integrator_Limit (New_Limit: in Fin_Deflections) is

   begin

      Tustin_Integrator.Update_Limit (New_Absolute_Limit => New_Limit);

   end Update_Integrator_Limit;
                                                                    pragma PAGE;
   procedure Update_Proportional_Gain
            (New_Proportional_Gain : in Acceleration_Command_Gains) is

   begin

      Pitch_Loop.Update_Proportional_Gain
                              (New_Proportional_Gain => New_Proportional_Gain);

   end Update_Proportional_Gain;

end Pitch_Autopilot;
```

3.3.5 NONGUIDANCE AND CONTROL

(This page intentionally left blank.)

3.3.5.1 AIR_DATA_PARTS (PACKAGE BODY) TLCSC P671 (CATALOG #P316-0)

This TLCSC contains parts which can be used to monitor air conditions.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.5.1.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
Compute_Outside_Air_Temperature	R228
Compute_Pressure_Ratio	R229
Compute_Mach	R230
Compute_Dynamic_Pressure	R231
Compute_Speed_Of_Sound	R232
Barometric_Altitude_Integration	R233

3.3.5.1.2 LOCAL ENTITIES DESIGN

None.

3.3.5.1.3 INPUT/OUTPUT

None.

3.3.5.1.4 LOCAL DATA

None.

3.3.5.1.5 PROCESS CONTROL

Not applicable.

3.3.5.1.6 PROCESSING

The following describes the processing performed by this part:

```
package body Air_Data_Parts is

   function Compute_Outside_Air_Temperature
               (Total_Temperature : Temperatures;
                Mach              : Mach_Numbers)
               return Temperatures is separate;

   function Compute_Pressure_Ratio
```

```
                (Measured_Static_Pressure    : Pressures;
                 Impact_Pressure             : Pressures;
                 Free_Stream_Static_Pressure : Pressures)
                return Ratios is separate;

   function Compute_Mach
                (Pressure_Ratio : Ratios)
                return Mach_Numbers is separate;

   function Compute_Dynamic_Pressure
                (Free_Stream_Static_Pressure : Pressures;
                 Mach                       : Mach_Numbers)
                return Pressures is separate;

   function Compute_Speed_of_Sound
                (Air_Temperature : Temperatures)
                return Velocities is separate;

    package body Barometric_Altitude_Integration is separate;

end Air_Data_Parts;
```

3.3.5.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.5.1.8 LIMITATIONS

None.

3.3.5.1.9 LLCSC DESIGN

3.3.5.1.9.1 BAROMETRIC_ALTITUDE_INTEGRATION (PACKAGE BODY) PACKAGE DESIGN (CATALOG #P322-0)

This unit is a generic package which computes barometric altitude by integration of the atmospheric equation of state.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.5.1.9.1.1 REQUIREMENTS ALLOCATION

This parts meets CAMP requirement R233.

3.3.5.1.9.1.2 LOCAL ENTITIES DESIGN

None.

3.3.5.1.9.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Temperatures	floating point type	Describes air temperatures
Pressures	floating point type	Describes pressure (i.e. weight per unit of area)
Distances	floating point type	Describes translational distances (e.g., Feet, Meters)
Molar_Gas_Constants	floating point type	Describes the type of the Gas Constant needed

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Gas_Constant	Molar_Gas_Constants	N/A	Constant which describes a standard gas constant
Maximum_Pressure_Change	Pressures	N/A	Maximum reasonable change expected in free stream static pressure between two measurement

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplies the Gas Constant by a Pressure yielding a Distance
"/"	function	Divides a Temperature by a Pressure, yielding a Pressure

3.3.5.1.9.1.4 LOCAL DATA

None.

3.3.5.1.9.1.5 PROCESS CONTROL

Not applicable.

3.3.5.1.9.1.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Air_Data_Parts)

package body Barometric_Altitude_Integration is

   Previous_Free_Stream_Static_Pressure : Pressures := Initial_Free_Stream_Pressure;
   Previous_Outside_Air_Temperature     : Temperatures := Initial_Temperature;
   Previous_Baro_Altitude               : Distances := Initial_Baro_Altitude;

end Barometric_Altitude_Integration;
```

3.3.5.1.9.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.5.1.9.1.8 LIMITATIONS

None.

3.3.5.1.9.1.9 LLCSC DESIGN

None.

3.3.5.1.9.1.10 UNIT DESIGN

3.3.5.1.9.1.10.1 COMPUTE_BAROMETRIC_ALTITUDE UNIT DESIGN

This unit is a function which computes barometric altitude by integration of the atmospheric equation of state.

3.3.5.1.9.1.10.1.1 REQUIREMENTS ALLOCATION

This parts meets CAMP requirement R233.

3.3.5.1.9.1.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.5.1.9.1.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Outside_Air_Temperature	Temperatures	in	Temperature of the air outside the missile
Free_Stream_Static_Pressure	Pressures	in	Measured static pressure corrected for errors
<returned value>	Distances	out	Altitude in feet based on the barometric pressure of the atmosphere

3.3.5.1.9.1.10.1.4 LOCAL DATA

None.

3.3.5.1.9.1.10.1.5 PROCESS CONTROL

Not applicable.

3.3.5.1.9.1.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
function Compute_Barometric_Altitude
           (Outside_Air_Temperature     : Temperatures;
            Free_Stream_Static_Pressure : Pressures)
           return Distances is

   Pressure_Change : Pressures;
   Baro_Altitude   : Distances;

begin

   Pressure_Change := Free_Stream_Static_Pressure -
                      Previous_Free_Stream_Static_Pressure;

   if (abs Pressure_Change) > Maximum_Pressure_Change then

      Previous_Free_Stream_Static_Pressure := Free_Stream_Static_Pressure;
      Baro_Altitude := Previous_Baro_Altitude;
```

```
      else

         Baro_Altitude :=
            Previous_Baro_Altitude -
            (0.5 * Gas_Constant) *
            (( (Outside_Air_Temperature / Free_Stream_Static_Pressure) +
               (Previous_Outside_Air_Temperature /
                Previous_Free_Stream_Static_Pressure)
              ) * Pressure_Change);

         Previous_Outside_Air_Temperature := Outside_Air_Temperature;
         Previous_Free_Stream_Static_Pressure := Free_Stream_Static_Pressure;
         Previous_Baro_Altitude := Baro_Altitude;

      end if;

      return Baro_Altitude;

   end Compute_Barometric_Altitude;
```

3.3.5.1.9.1.10.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.5.1.9.1.10.1.8 LIMITATIONS

None.

3.3.5.1.10 UNIT DESIGN

3.3.5.1.10.1 COMPUTE_OUTSIDE_AIR_TEMPERATURE (FUNCTION BODY) UNIT DESIGN (CATALOG #P317-0)

This unit is a generic function which computes air temperature outside of a missile.

3.3.5.1.10.1.1 REQUIREMENTS ALLOCATION

This parts meets CAMP Requirement R228

3.3.5.1.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.5.1.10.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Temperatures	floating point type	Describes air temperatures
Mach_Numbers	floating point type	Describes air speed as a ratio of the speed of sound
Real	floating point type	General floating point type

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Recovery_Factor	Real	N/A	Constant for computing Air Temp

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplies a Real by a Mach Number, yielding a Mach Number
"/"	function	Divides a Temperature by a Mach Number, yielding a Temperature

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Total_Temperature	Temperatures	in	Air temperature measured by the air data instruments
Mach	Mach_Numbers	in	Missile airspeed as a fraction of the speed of sound
<returned value>	Temperatures	out	Temperature of the air outside of the missile

3.3.5.1.10.1.4 LOCAL DATA

None.

3.3.5.1.10.1.5 PROCESS CONTROL

Not applicable.

3.3.5.1.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Air_Data_Parts)
function Compute_Outside_Air_Temperature
            (Total_Temperature : Temperatures;
             Mach              : Mach_Numbers)
            return Temperatures is

begin

   return Total_Temperature / (1.0 + 0.2 * Recovery_Factor * Mach * Mach);

end Compute_Outside_Air_Temperature;
```

3.3.5.1.10.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.5.1.10.1.8 LIMITATIONS

None.

3.3.5.1.10.2 COMPUTE_PRESSURE_RATIO (FUNCTION BODY) UNIT DESIGN (CATALOG #P318-0)

This unit is a generic function which computes pressure ratio from measured static pressure, measured impact pressure, and free stream static pressure.

3.3.5.1.10.2.1 REQUIREMENTS ALLOCATION

This parts meets CAMP requirement R229

3.3.5.1.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.5.1.10.2.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Pressures	floating point type	Describes pressure (i.e. weight per unit of area)
Ratios	floating point type	A unitless floating point type describing ratio of one pressure to another

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"/"	function	Divides a Pressure by a Pressure, yielding a ratio

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Measured_Static_Pressure	Pressures	in	Static pressure measured by the air data system
Impact_Pressure	Pressures	in	Measured difference between totl pressure and static pressure
Free_Stream_Static_Pressure	Pressures	in	Measured static pressure which has been corrected for errors
<returned value>	Ratios	out	Unitless quantity computed from static and impact pressure

3.3.5.1.10.2.4 LOCAL DATA

None.

3.3.5.1.10.2.5 PROCESS CONTROL

Not applicable.

3.3.5.1.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Air_Data_Parts)
function Compute_Pressure_Ratio
            (Measured_Static_Pressure    : Pressures;
             Impact_Pressure             : Pressures;
             Free_Stream_Static_Pressure : Pressures)
            return Ratios is

begin

   return (Measured_Static_Pressure + Impact_Pressure) /
          Free_Stream_Static_Pressure;

end Compute_Pressure_Ratio;
```

3.3.5.1.10.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.5.1.10.2.8 LIMITATIONS

None.

3.3.5.1.10.3 COMPUTE_MACH (FUNCTION BODY) UNIT DESIGN (CATALOG #P319-0)

This unit is a generic function which computes missile mach given pressure ratio.

3.3.5.1.10.3.1 REQUIREMENTS ALLOCATION

This parts meets CAMP requirement R230.

3.3.5.1.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.5.1.10.3.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Mach_Numbers	floating point type	Describes air speed as a ratio of the speed of sound
Ratios	floating point type	A unitless floating point type describing ratio of one pressure to another

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
C0	Ratios		First curve fit parameter
C1	Ratios		Second curve fit parameter
C2	Ratios		Third curve fit parameter

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
Sqrt	function	Computes the square root of Ratio, yielding a Mach Number

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Pressure_Ratio	Ratios	in	Unitless quantity computed from static and impact pressures
<returned value>	Mach_Numbers	out	Missile airspeed as a fraction of the speed of sound

3.3.5.1.10.3.4 LOCAL DATA

None.

3.3.5.1.10.3.5 PROCESS CONTROL

Not applicable.

3.3.5.1.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Air_Data_Parts)
function Compute_Mach (Pressure_Ratio : Ratios) return Mach_Numbers is

begin

   return Sqrt (C0 + Pressure_Ratio * (C1 + C2 * Pressure_Ratio));

end Compute_Mach;
```

3.3.5.1.10.3.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.5.1.10.3.8 LIMITATIONS

None.

3.3.5.1.10.4 COMPUTE_DYNAMIC_PRESSURE (FUNCTION BODY) UNIT DESIGN (CATALOG #P320-0)

This unit is a generic function which computes dynamic pressure from missile mach number and free stream static pressure.

3.3.5.1.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP Requirement R231.

3.3.5.1.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.5.1.10.4.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Pressures	floating point type	Describes pressure (i.e. weight per unit of area)
Mach_Numbers	floating point type	Describes air speed as a ratio of the speed of sound

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplies a Pressure by a Mach Number, yielding a Pressure

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Free_Stream_Static_Pressure	Pressures	in	Measured static pressure which has been corrected for errors
Mach	Mach_Numbers	in	Missile airspeed as a fraction of the speed of sound
<returned value>	Pressures	out	Missile dynamic pressure

3.3.5.1.10.4.4 LOCAL DATA

None.

3.3.5.1.10.4.5 PROCESS CONTROL

Not applicable.

3.3.5.1.10.4.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Air_Data_Parts)

function Compute_Dynamic_Pressure
            (Free_Stream_Static_Pressure : Pressures;
             Mach                       : Mach_Numbers)
            return Pressures is

begin

   return 0.7 * Free_Stream_Static_Pressure * (Mach * Mach);

end Compute_Dynamic_Pressure;
```

3.3.5.1.10.4.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.5.1.10.4.8 LIMITATIONS

None.

3.3.5.1.10.5 COMPUTE_SPEED_OF_SOUND (FUNCTION BODY) UNIT DESIGN (CATALOG #P321-0)

This unit is a generic function which computes the speed of sound given the temperature of the air.

3.3.5.1.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R232.

3.3.5.1.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.5.1.10.5.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Temperatures	floating point type	Describes air temperatures
Velocities	floating point type	Describes air speed

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Value	Description
Speed_Of_Sound_Constant	Velocities	N/A	Standard speed of sound at sea level

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplies a Velocity by a Temperature, yielding a Velocity
Sqrt	function	Computes the square root of a Temperature, yielding a Temperature

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Air_Temperature	Temperatures	in	Temperature of the air
<returned value>	Velocities	out	Speed of sound in air

3.3.5.1.10.5.4 LOCAL DATA

None.

3.3.5.1.10.5.5 PROCESS CONTROL

Not applicable.

3.3.5.1.10.5.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Air_Data_Parts)

function Compute_Speed_of_Sound (Air_Temperature : Temperatures)
                                 return Velocities is

begin

   return Speed_Of_Sound_Constant * Sqrt (Air_Temperature);

end Compute_Speed_of_Sound;
```

3.3.5.1.10.5.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.5.1.10.5.8 LIMITATIONS

None.

```
package body Air_Data_Parts is

   function Compute_Outside_Air_Temperature
               (Total_Temperature : Temperatures;
                Mach              : Mach_Numbers)
               return Temperatures is separate;

   function Compute_Pressure_Ratio
               (Measured_Static_Pressure    : Pressures;
                Impact_Pressure             : Pressures;
                Free_Stream_Static_Pressure : Pressures)
               return Ratios is separate;

   function Compute_Mach
               (Pressure_Ratio : Ratios)
               return Mach_Numbers is separate;

   function Compute_Dynamic_Pressure
               (Free_Stream_Static_Pressure : Pressures;
                Mach                       : Mach_Numbers)
               return Pressures is separate;

   function Compute_Speed_Of_Sound
               (Air_Temperature : Temperatures)
               return Velocities is separate;

    package body Barometric_Altitude_Integration is separate;

end Air_Data_Parts;
```

```
separate (Air_Data_Parts)
function Compute_Outside_Air_Temperature
            (Total_Temperature : Temperatures;
             Mach              : Mach_Numbers)
            return Temperatures is

begin

   return Total_Temperature / (1.0 + 0.2 * Recovery_Factor * Mach * Mach);

end Compute_Outside_Air_Temperature;
```

```
separate (Air_Data_Parts)
function Compute_Pressure_Ratio
            (Measured_Static_Pressure    : Pressures;
             Impact_Pressure             : Pressures;
             Free_Stream_Static_Pressure : Pressures)
            return Ratios is

begin

   return (Measured_Static_Pressure + Impact_Pressure) /
          Free_Stream_Static_Pressure;

end Compute_Pressure_Ratio;
```

```
separate (Air_Data_Parts)
function Compute_Mach (Pressure_Ratio : Ratios) return Mach_Numbers is

begin

   return Sqrt (C0 + Pressure_Ratio * (C1 + C2 * Pressure_Ratio));

end Compute_Mach;
```

```
separate (Air_Data_Parts)

function Compute_Dynamic_Pressure
            (Free_Stream_Static_Pressure : Pressures;
             Mach                       : Mach_Numbers)
            return Pressures is

begin

   return 0.7 * Free_Stream_Static_Pressure * (Mach * Mach);

end Compute_Dynamic_Pressure;
```

```
separate (Air_Data_Parts)

function Compute_Speed_Of_Sound (Air_Temperature : Temperatures)
                                 return Velocities is

begin

   return Speed_Of_Sound_Constant * Sqrt (Air_Temperature);

end Compute_Speed_Of_Sound;
```

```
separate (Air_Data_Parts)

package body Barometric_Altitude_Integration is

   Previous_Free_Stream_Static_Pressure : Pressures := Initial_Free_Stream_Pressure;
   Previous_Outside_Air_Temperature     : Temperatures := Initial_Temperature;
   Previous_Baro_Altitude               : Distances := Initial_Baro_Altitude;
                                                                    pragma PAGE;
   function Compute_Barometric_Altitude
               (Outside_Air_Temperature     : Temperatures;
                Free_Stream_Static_Pressure : Pressures)
               return Distances is

      Pressure_Change : Pressures;
      Baro_Altitude   : Distances;

   begin

      Pressure_Change := Free_Stream_Static_Pressure -
                         Previous_Free_Stream_Static_Pressure;

      if (abs Pressure_Change) > Maximum_Pressure_Change then

         Previous_Free_Stream_Static_Pressure := Free_Stream_Static_Pressure;
         Baro_Altitude := Previous_Baro_Altitude;

      else

         Baro_Altitude :=
            Previous_Baro_Altitude -
            (0.5 * Gas_Constant) *
            (( (Outside_Air_Temperature / Free_Stream_Static_Pressure) +
               (Previous_Outside_Air_Temperature /
                Previous_Free_Stream_Static_Pressure)
              ) * Pressure_Change);

         Previous_Outside_Air_Temperature := Outside_Air_Temperature;
         Previous_Free_Stream_Static_Pressure := Free_Stream_Static_Pressure;
         Previous_Baro_Altitude := Baro_Altitude;

      end if;

      return Baro_Altitude;

   end Compute_Barometric_Altitude;

end Barometric_Altitude_Integration;
```

(This page left intentionally blank.)

3.3.5.2 FUEL_CONTROL_PARTS TLCSC P672 (CATALOG #P1096-0)

This TLCSC contains parts which can be used to manage missile fuel consumption.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.5.2.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this part:

Name	Requirements Allocation
Throttle_Command_Manager	R234

3.3.5.2.2 LOCAL ENTITIES DESIGN

None.

3.3.5.2.3 INPUT/OUTPUT

None.

3.3.5.2.4 LOCAL DATA

None.

3.3.5.2.5 PROCESS CONTROL

Not applicable.

3.3.5.2.6 PROCESSING

The following describes the processing performed by this part:

```
with Signal_Processing;
with Autopilot;

package body Fuel_Control_Parts is

end Fuel_Control_Parts;
```

3.3.5.2.7 UTILIZATION OF OTHER ELEMENTS

The following library units are with'd by this part:

1. Signal_Processing
2. Autopilot

3.3.5.2.8 LIMITATIONS

None.

3.3.5.2.9 LLCSC DESIGN

3.3.5.2.9.1 THROTTLE_COMMAND_MANAGER PACKAGE DESIGN

This LLCSC is a generic package which manages the throttle command.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.5.2.9.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP Requirement R234.

3.3.5.2.9.1.2 LOCAL ENTITIES DESIGN

Packages:

The packages Integral_Plus_Proportional_Gain, Tustin_Integrator_With_Limit, Tustin_Integrator_With_Asymmetric_Limit, and Absolute_Limiter are instantiated inside the package body.

Subprograms:

This package contains a sequence of statements which are executed when this part is elaborated. This code initializes the state of the throttle command manager.

3.3.5.2.9.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Mach_Numbers	floating point type	Represents missile speed as a ratio of the speed of sound
Mach_Number_Gains	floating point type	Represents a gain which converts from Mach Number to Throttle Command
Throttle_Commands	floating point type	Represents a command to open/close the throttle

Data objects:

The following table describes the generic formal objects required by this part:

Name	Type	Mode	Description
Initial_Mach_Command	Mach_Numbers	in	Mach Number of missile at startup
Initial_Mach_Feedback	Mach_Numbers	in	Mach Feedback from missile at startup
Initial_Mach_Error_Limit	Mach_Numbers	in	Limit of Mach Error
Initial_Mach_Error_Gain	Mach_Number_Gain	in	Gain to convert from mach error to raw throttle command
Initial_Mach_Error_Integral_Limit	Throttle_Commands	in	Limit for Mach Error Integral at startup
Initial_Throttle_Command	Throttle_Commands	in	Throttle Command at startup
Initial_Throttle_Command_Rate_Limit	Throttle_Commands	in	Limit on Throttle Command Rate at startup
Initial_Lower_Throttle_Command_Limit	Throttle_Commands	in	Lower Limit of Throttle Command
Initial_Upper_Throttle_Command_Limit	Throttle_Commands	in	Upper Limit of Throttle Command
Initial_Throttle_Bandwidth	Throttle_Commands	in	3 db bandwidth of the throttle command

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplies a Mach_Number by a Mach_Number_Gain yielding a Throttle_Command

3.3.5.2.9.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Mach_Error	Mach_Numbers	Difference between Measured Mach and requested Mach
Throttle_Command_Rate	Throttle_Commands	Rate at which the throttle is being opened or closed
Stored_Throttle_Command	Throttle_Commands	Previous Throttle Command
Raw_Throttle_Command	Throttle_Commands	Computed Throttle Command (not limited)
Initial_IPP_Integral	Throttle_Commands	The initial state computed and sent to the Integral Plus Proporational Gain package

3.3.5.2.9.1.5 PROCESS CONTROL

Not applicable.

3.3.5.2.9.1.6 PROCESSING

The following describes the processing performed by this part:

```
package body Throttle_Command_Manager is

   Mach_Error                : Mach_Numbers;
   Throttle_Command_Rate     : Throttle_Commands;
   Stored_Throttle_Command   : Throttle_Commands;
   Raw_Throttle_Command      : Throttle_Commands;
   Initial_IPP_Integral      : Throttle_Commands;


   package Mach_Error_Limiter is new Signal_Processing.
                                     Absolute_Limiter
           (Signal_Type            => Mach_Numbers,
            Initial_Absolute_Limit => Initial_Mach_Error_Limit);


   package Throttle_Command_Rate_Limiter is new Signal_Processing.
                                                Absolute_Limiter
   (Signal_Type            => Throttle_Commands,
    Initial_Absolute_Limit => Initial_Throttle_Command_Rate_Limit);
```

```
      package IPP_Tustin is new Signal_Processing.
                                Tustin_Integrator_With_Limit
          (Signals              => Mach_Numbers,
           States               => Throttle_Commands,
           Gained_Signals       => Throttle_Commands,
           Gains                => Mach_Number_Gains,
           Initial_Tustin_Gain  => Initial_Mach_Error_Gain,
           Initial_Signal_Level => Initial_Mach_Command,
           Initial_State        => Initial_Throttle_Command,
           Initial_Signal_Limit => Initial_Mach_Error_Integral_Limit);

      package IPP_Gain is new Autopilot.Integral_Plus_Proportional_Gain
                (Input_Signals             => Mach_Numbers,
                 Gains                     => Mach_Number_Gains,
                 Integrated_Signals        => Throttle_Commands,
                 Initial_Proportional_Gain => Initial_Mach_Error_Gain,
                 Tustin_Integrate          => IPP_Tustin.Integrate);


      package Tustin is new Signal_Processing.
                           Tustin_Integrator_With_Asymmetric_Limit
             (Signals                    => Throttle_Commands,
              States                     => Throttle_Commands,
              Gained_Signals             => Throttle_Commands,
              Gains                      => Throttle_Commands,
              Initial_Tustin_Gain        => Initial_Throttle_Bandwidth,
              Initial_Signal_Level       => Initial_Throttle_Command,
              Initial_State              => Initial_Throttle_Command,
              Initial_Signal_Lower_Limit =>
                               Initial_Lower_Throttle_Command_Limit,
              Initial_Signal_Upper_Limit =>
                               Initial_Upper_Throttle_Command_Limit);


-- -------------------------------
-- --begin processing for Throttle_
-- --Command_Manager package body
-- -------------------------------

   begin

      Mach_Error := 0.0;

      Mach_Error:= Mach_Error_Limiter.Limit (Mach_Error);

      Raw_Throttle_Command    := Initial_Throttle_Command;
      Stored_Throttle_Command := 0.0;

      Initial_IPP_Integral := Raw_Throttle_Command -
                              Mach_Error * Initial_Mach_Error_Gain;

      IPP_Tustin.Reset (Integrator_State => Initial_IPP_Integral,
                        Signal           => Mach_Error);

   end Throttle_Command_Manager;
```

3.3.5.2.9.1.7 UTILIZATION OF OTHER ELEMENTS

The following library units are with'd by this part's TLCSC:
1. Signal Processing
2. Autopilot

UTILIZATION OF EXTERNAL ELEMENTS:

Packages:

The following table summarizes the external packages required by this part:

Name	Type	Source	Description
Absolute_Limiter	generic package	(1)	Limits a value by an absolute value
Tustin_Integrator_With_Limit	generic package	(1)	Performs a Tustin integration of an independent variable and performs an absolute limit of the result
Tustin_Integrator_With_Asymmetric_	generic package	(1)	Performs a Tustin integration of an independent variable and performs an upper/lower limit of the result
Integral_Plus_Proportional_Gain	generic package	(2)	Performs an integral plus proportional gain computation of a subject variable

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Limit	function	Mach_Error_Limiter	Limits the Mach Error
Reset	procedure	IPP_Tustin	Initializes the Tustin Integrator used by the IPP Gain

Data types:

The following table summarizes the types required by this part and defined elsewhere in the LLCSC Package Specification:

Name	Type	Description
Mach_Numbers	floating point type	Represents missile speed as a ratio of the speed of sound
Mach_Number_Gains	floating point type	Represents a gain which converts from Mach Number to Throttle Command
Throttle_Commands	floating point type	Represents a command to open/close the throttle

Data objects:

The following table summarizes the objects required by this part and defined in the LLCSC package body:

Name	Type	Description
Mach_Error	Mach_Numbers	Difference between Measured Mach and requested Mach
Stored_Throttle_Command	Throttle_Commands	Previous Throttle Command
Raw_Throttle_Command	Throttle_Commands	Computed Throttle Command (not limited)

3.3.5.2.9.1.8 LIMITATIONS

None.

3.3.5.2.9.1.9 LLCSC DESIGN

None.

3.3.5.2.9.1.10 UNIT DESIGN

3.3.5.2.9.1.10.1 COMPUTE_THROTTLE_COMMAND UNIT DESIGN

This unit is a function which computes the new throttle command.

3.3.5.2.9.1.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R234.

3.3.5.2.9.1.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.5.2.9.1.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Mach_Command	Mach_Numbers	in	Wanted Missile mach
Mach_Feedback	Mach_Numbers	in	Measured Missile mach
<returned value>	Throttle_Commands	in	Computed Throttle Command

3.3.5.2.9.1.10.1.4 LOCAL DATA

None.

3.3.5.2.9.1.10.1.5 PROCESS CONTROL

Not applicable.

3.3.5.2.9.1.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
function Compute_Throttle_Command
            (Mach_Command                : in Mach_Numbers;
             Mach_Feedback               : in Mach_Numbers)
            return Throttle_Commands is
begin

   Mach_Error := Mach_Command - Mach_Feedback;

   Mach_Error:= Mach_Error_Limiter.Limit (Mach_Error);

   Raw_Throttle_Command := IPP_Gain.Integrate (Signal => Mach_Error);

   Throttle_Command_Rate := Raw_Throttle_Command -
                            Stored_Throttle_Command;

   Throttle_Command_Rate := Throttle_Command_Rate_Limiter.Limit
                            (Throttle_Command_Rate);
```

```
        Stored_Throttle_Command := Tustin.Integrate
                                   (Signal => Throttle_Command_Rate);

        return Stored_Throttle_Command;

    end Compute_Throttle_Command;
```

3.3.5.2.9.1.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Limit	function	Mach_Error_Limiter	Limits the Mach Error
Limit	function	Throttle_Command_Rate_Limiter	Limits the Throttle Command Rate
Integrate	function	IPP_Gain	Computes the Raw Throttle Command
Integrate	function	Tustin_Integrator	Computes the Final Throttle Command

Data types:

The following table summarizes the types required by this part and defined elsewhere in the LLCSC Package Specification:

Name	Type	Description
Mach_Numbers	floating point type	Represents missile speed as a ratio of the speed of sound
Throttle_Commands	floating point type	Represents a command to open/close the throttle

Data objects:

The following table summarizes the objects required by this part and defined in the LLCSC package body:

Name	Type	Description
Mach_Error	Mach_Numbers	Difference between Measured Mach and requested Mach
Throttle_Command_Rate	Throttle_Commands	Rate at which the throttle is being opened or closed
Stored_Throttle_Command	Throttle_Commands	Previous Throttle Command
Raw_Throttle_Command	Throttle_Commands	Computed Throttle Command (not limited)

3.3.5.2.9.1.10.1.8 LIMITATIONS

None.

3.3.5.2.9.1.10.2 UPDATE_MACH_ERROR_LIMIT UNIT DESIGN

This unit is a procedure which updates the mach error limit.

3.3.5.2.9.1.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R234.

3.3.5.2.9.1.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.5.2.9.1.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
New_Limit	Mach_Numbers	in	New Mach Error limit

3.3.5.2.9.1.10.2.4 LOCAL DATA

None.

3.3.5.2.9.1.10.2.5 PROCESS CONTROL

Not applicable.

3.3.5.2.9.1.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Mach_Error_Limit (New_Limit : in Mach_Numbers) is
begin
   Mach_Error_Limiter.Update_Limit (New_Absolute_Limit => New_Limit);
end Update_Mach_Error_Limit;
```

3.3.5.2.9.1.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Update_Limit	function	Mach_Error_ Limiter	Updates the mach error limit

Data types:

The following table summarizes the types required by this part and defined elsewhere in the LLCSC Package Specification:

Name	Type	Description
Mach_Numbers	floating point type	Represents missile speed as a ratio of the speed of sound

3.3.5.2.9.1.10.2.8 LIMITATIONS

None.

3.3.5.2.9.1.10.3 UPDATE_MACH_ERROR_INTEGRAL_LIMIT UNIT DESIGN

This unit is a procedure which updates the mach error integral limit.

3.3.5.2.9.1.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R234.

3.3.5.2.9.1.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.5.2.9.1.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
New_Limit	Mach_ Numbers	in	New Mach Error integral limit

3.3.5.2.9.1.10.3.4 LOCAL DATA

None.

3.3.5.2.9.1.10.3.5 PROCESS CONTROL

Not applicable.

3.3.5.2.9.1.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Mach_Error_Integral_Limit
                                (New_Limit : in Throttle_Commands) is
begin
   IPP_Tustin.Update_Limit (New_Absolute_Limit => New_Limit);
end Update_Mach_Error_Integral_Limit;
```

3.3.5.2.9.1.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Update_Limit	function	Mach_Error_Integral_Limiter	Updates the mach error integral limit

Data types:

The following table summarizes the types required by this part and defined elsewhere in the LLCSC Package Specification:

Name	Type	Description
Mach_Numbers	floating point type	Represents missile speed as a ratio of the speed of sound

3.3.5.2.9.1.10.3.8 LIMITATIONS

None.

3.3.5.2.9.1.10.4 UPDATE_THROTTLE_RATE_LIMIT UNIT DESIGN

This unit is a procedure which updates the throttle rate limit.

3.3.5.2.9.1.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R234.

3.3.5.2.9.1.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.5.2.9.1.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
New_Limit	Throttle_Commands	in	New Throttle Rate limit

3.3.5.2.9.1.10.4.4 LOCAL DATA

None.

3.3.5.2.9.1.10.4.5 PROCESS CONTROL

Not applicable.

3.3.5.2.9.1.10.4.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Throttle_Rate_Limit(New_Limit : in Throttle_Commands) is
begin
   Throttle_Command_Rate_Limiter.Update_Limit
                                (New_Absolute_Limit => New_Limit);
end Update_Throttle_Rate_Limit;
```

3.3.5.2.9.1.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Update_Limit	function	Throttle_Command_Rate Limiter	Updates the Throttle Command Rate limit

Data types:

The following table summarizes the types required by this part and defined elsewhere in the LLCSC Package Specification:

Name	Type	Description
Throttle_Commands	floating point type	Represents a command to open/close the throttle

3.3.5.2.9.1.10.4.8 LIMITATIONS

None.

3.3.5.2.9.1.10.5 UPDATE_THROTTLE_COMMAND_LIMITS UNIT DESIGN

This unit is a procedure which updates the throttle command limits.

3.3.5.2.9.1.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R234.

3.3.5.2.9.1.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.5.2.9.1.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
New_Lower_Limit	Throttle_Commands	in	New Lower Throttle Command Limit
New_Upper_Limit	Throttle_Commands	in	New Upper Throttle Command Limit

3.3.5.2.9.1.10.5.4 LOCAL DATA

None.

3.3.5.2.9.1.10.5.5 PROCESS CONTROL

Not applicable.

3.3.5.2.9.1.10.5.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Throttle_Command_Limits
                          (New_Lower_Limit : in Throttle_Commands;
                           New_Upper_Limit : in Throttle_Commands) is
begin
   Tustin.Update_Limits (New_Lower_Limit => New_Lower_Limit,
                         New_Upper_Limit => New_Upper_Limit);
```

```
end Update_Throttle_Command_Limits;
```

3.3.5.2.9.1.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Update_Limit	function	Tustin	Updates the Throttle Command limits

Data types:

The following table summarizes the types required by this part and defined elsewhere in the LLCSC Package Specification:

Name	Type	Description
Throttle_Commands	floating point type	Represents a command to open/close the throttle

3.3.5.2.9.1.10.5.8 LIMITATIONS

None.

3.3.5.2.9.1.10.6 UPDATE_MACH_ERROR_GAIN UNIT DESIGN

This unit is a procedure which updates the mach error gain

3.3.5.2.9.1.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R234.

3.3.5.2.9.1.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.5.2.9.1.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
New_Gain	Mach_Numbers	in	New Mach Error Gain

3.3.5.2.9.1.10.6.4 LOCAL DATA

None.

3.3.5.2.9.1.10.6.5 PROCESS CONTROL

Not applicable.

3.3.5.2.9.1.10.6.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Mach_Error_Gain (New_Gain : in Mach_Number_Gains) is
begin
   IPP_Tustin.Update_Gain (New_Gain => New_Gain);
end Update_Mach_Error_Gain;
```

3.3.5.2.9.1.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Update_Gain	procedure	IPP_Tustin	Updates the Mach Error Gain

Data types:

The following table summarizes the types required by this part and defined elsewhere in the LLCSC Package Specification:

Name	Type	Description
Mach_Numbers	floating point type	Represents missile speed as a ratio of the speed of sound

3.3.5.2.9.1.10.6.8 LIMITATIONS

None.

3.3.5.2.9.1.10.7 UPDATE_THROTTLE_BANDWIDTH UNIT DESIGN

This unit is a procedure which updates the 3 db throttle command bandwidth

3.3.5.2.9.1.10.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R234.

3.3.5.2.9.1.10.7.2 LOCAL ENTITIES DESIGN

None.

3.3.5.2.9.1.10.7.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
New_Bandwidth	Throttle_Commands	in	New Throttle Bandwidth

3.3.5.2.9.1.10.7.4 LOCAL DATA

None.

3.3.5.2.9.1.10.7.5 PROCESS CONTROL

Not applicable.

3.3.5.2.9.1.10.7.6 PROCESSING

The following describes the processing performed by this part:

```
procedure Update_Throttle_Bandwidth
```

```
                              (New_Bandwidth : in Throttle_Commands) is
      begin
         Tustin.Update_Gain (New_Gain => New_Bandwidth);
      end Update_Throttle_Bandwidth;
```

3.3.5.2.9.1.10.7.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined elsewhere in the parent top level component:

Name	Type	Source	Description
Update_Gain	procedure	Tustin	Updates the Gain (i.e., throttle command bandwidth)

Data types:

The following table summarizes the types required by this part and defined elsewhere in the LLCSC Package Specification:

Name	Type	Description
Throttle_Commands	floating point type	Represents a command to open/close the throttle

3.3.5.2.9.1.10.7.8 LIMITATIONS

None.

3.3.5.2.10 UNIT DESIGN

None.

(This page left intentionally blank.)

```
with Signal_Processing;
with Autopilot;

package body Fuel_Control_Parts is

                                                          pragma PAGE;
   package body Throttle_Command_Manager is

      Mach_Error                 : Mach_Numbers;
      Throttle_Command_Rate      : Throttle_Commands;
      Stored_Throttle_Command    : Throttle_Commands;
      Raw_Throttle_Command       : Throttle_Commands;
      Initial_Ipp_Integral       : Throttle_Commands;


      package Mach_Error_Limiter is new Signal_Processing.
                                        Absolute_Limiter
                (Signal_Type            => Mach_Numbers,
                 Initial_Absolute_Limit => Initial_Mach_Error_Limit);


      package Throttle_Command_Rate_Limiter is new Signal_Processing.
                                                   Absolute_Limiter
      (Signal_Type            => Throttle_Commands,
       Initial_Absolute_Limit => Initial_Throttle_Command_Rate_Limit);



      package Ipp_Tustin is new Signal_Processing.
                                Tustin_Integrator_With_Limit
          (Signals             => Mach_Numbers,
           States              => Throttle_Commands,
           Gained_Signals      => Throttle_Commands,
           Gains               => Mach_Number_Gains,
           Initial_Tustin_Gain => Initial_Mach_Error_Gain,
           Initial_Signal_Level => Initial_Mach_Command,
           Initial_State       => Initial_Throttle_Command,
           Initial_Signal_Limit => Initial_Mach_Error_Integral_Limit);

      package Ipp_Gain is new Autopilot.Integral_Plus_Proportional_Gain
                 (Input_Signals             => Mach_Numbers,
                  Gains                     => Mach_Number_Gains,
                  Integrated_Signals        => Throttle_Commands,
                  Initial_Proportional_Gain => Initial_Mach_Error_Gain,
                  Tustin_Integrate          => Ipp_Tustin.Integrate);


      package Tustin is new Signal_Processing.
                            Tustin_Integrator_With_Asymmetric_Limit
             (Signals                   => Throttle_Commands,
              States                    => Throttle_Commands,
              Gained_Signals            => Throttle_Commands,
              Gains                     => Throttle_Commands,
              Initial_Tustin_Gain       => Initial_Throttle_Bandwidth,
              Initial_Signal_Level      => Initial_Throttle_Command,
              Initial_State             => Initial_Throttle_Command,
```

```
         Initial_Signal_Lower_Limit =>
                        Initial_Lower_Throttle_Command_Limit,
         Initial_Signal_Upper_Limit =>
                        Initial_Upper_Throttle_Command_Limit);
                                                              pragma PAGE;
function Compute_Throttle_Command
            (Mach_Command                 : in Mach_Numbers;
             Mach_Feedback                : in Mach_Numbers)
            return Throttle_Commands is
begin

   Mach_Error :=  Mach_Command - Mach_Feedback;

   Mach_Error:= Mach_Error_Limiter.Limit (Mach_Error);

   Raw_Throttle_Command := Ipp_Gain.Integrate (Signal => Mach_Error);

   Throttle_Command_Rate := Raw_Throttle_Command -
                            Stored_Throttle_Command;

   Throttle_Command_Rate := Throttle_Command_Rate_Limiter.Limit
                            (Throttle_Command_Rate);

   Stored_Throttle_Command := Tustin.Integrate
                              (Signal => Throttle_Command_Rate);

   return Stored_Throttle_Command;

end Compute_Throttle_Command;
                                                              pragma PAGE;
procedure Update_Mach_Error_Limit (New_Limit : in Mach_Numbers) is
begin
   Mach_Error_Limiter.Update_Limit (New_Absolute_Limit => New_Limit);
end Update_Mach_Error_Limit;
                                                              pragma PAGE;
procedure Update_Mach_Error_Integral_Limit
                              (New_Limit : in Throttle_Commands) is
begin
   Ipp_Tustin.Update_Limit (New_Absolute_Limit => New_Limit);
end Update_Mach_Error_Integral_Limit;
                                                              pragma PAGE;
procedure Update_Throttle_Rate_Limit(New_Limit : in Throttle_Commands) is
begin
   Throttle_Command_Rate_Limiter.Update_Limit
                               (New_Absolute_Limit => New_Limit);
end Update_Throttle_Rate_Limit;
                                                              pragma PAGE;
procedure Update_Throttle_Command_Limits
                            (New_Lower_Limit : in Throttle_Commands;
                             New_Upper_Limit : in Throttle_Commands) is
begin
   Tustin.Update_Limits (New_Lower_Limit => New_Lower_Limit,
                         New_Upper_Limit => New_Upper_Limit);
end Update_Throttle_Command_Limits;
                                                              pragma PAGE;
procedure Update_Mach_Error_Gain (New_Gain : in Mach_Number_Gains) is
begin
```

```
         Ipp_Tustin.Update_Gain (New_Gain => New_Gain);
      end Update_Mach_Error_Gain;
                                                                  pragma PAGE;
      procedure Update_Throttle_Bandwidth
                              (New_Bandwidth : in Throttle_Commands) is
      begin
         Tustin.Update_Gain (New_Gain => New_Bandwidth);
      end Update_Throttle_Bandwidth;

                                                                  pragma PAGE;
-- --------------------------------
-- --begin processing for Throttle_
-- --Command_Manager package body
-- --------------------------------

   begin

      Mach_Error := 0.0;

      Mach_Error:= Mach_Error_Limiter.Limit (Mach_Error);

      Raw_Throttle_Command    := Initial_Throttle_Command;
      Stored_Throttle_Command := 0.0;

      Initial_Ipp_Integral := Raw_Throttle_Command -
                              Mach_Error * Initial_Mach_Error_Gain;

      Ipp_Tustin.RESET (Integrator_State => Initial_Ipp_Integral,
                        Signal           => Mach_Error);

   end Throttle_Command_Manager;

end Fuel_Control_Parts;
```

(This page left intentionally blank.)

3.3.6 MATHEMATICAL

(This page intentionally left blank.)

3.3.6.1 COORDINATE VECTOR MATRIX ALGEBRA (BODY) TLCSC P681 (CATALOG #P53-0)

This part consists of generic packages and functions which define and/or operate on coordinate vectors and matrices. A coordinate vector is a three-element array. A coordinate matrix is a 3 x 3 array. These arrays are dimensioned with scalar types defined by the user.

WARNING: The units in this part ASSUME the axes types used to dimension the arrays have a length of 3. If they do not, the units will not function properly. No length checks are performed by the units.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.1.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this TLCSC:

Name	Type	Requirements Allocation
Vector_Operations	generic package	R024, R050, R051, R052
Matrix_Operations	generic package	R070, R071, R060, R067, R072 R078
Vector_Scalar_Operations	generic package	R054, R055
Matrix_Scalar_Operations	generic package	R056, R057
Cross_Product	generic function	R053
Matrix_Vector_Multiply	generic function	R049
Matrix_Matrix_Multiply	generic function	R068

3.3.6.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.3 INPUT/OUTPUT

None.

3.3.6.1.4 LOCAL DATA

None.

3.3.6.1.5 PROCESS CONTROL

Not applicable.

3.3.6.1.6 PROCESSING

The following describes the processing performed by this part:

```
with General_Purpose_Math;
package body Coordinate_Vector_Matrix_Algebra is

   package body Matrix_Operations is separate;

   package body Matrix_Scalar_Operations is separate;

   function Cross_Product (Left  : Left_Vectors;
                           Right : Right_Vectors)
                          return Result_Vectors is separate;

   function Matrix_Vector_Multiply (Matrix : Matrices;
                                    Vector : Input_Vectors)
                                   return Output_Vectors is separate;

   function Matrix_Matrix_Multiply (Matrix1 : Left_Matrices;
                                    Matrix2 : Right_Matrices)
                                   return Result_Matrices is separate;

end Coordinate_Vector_Matrix_Algebra;
```

3.3.6.1.7 UTILIZATION OF OTHER ELEMENTS

The following library units are with'd by this part:
1. General_Purpose_Math (GPMath)

3.3.6.1.8 LIMITATIONS

None.

3.3.6.1.9 LLCSC DESIGN

3.3.6.1.9.1 VECTOR OPERATIONS PACKAGE DESIGN (CATALOG #P54-0)

This part, which is a package body, provides general operations on three-element, coordinate vectors.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.1.9.1.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of CAMP requirements to this LLCSC.

Name	Type	Requirement Allocation
"+"	function	R050
"-"	function	R051
Dot_Product	function	R052
Length	function	R208
Sparse_Right_Z_Add	function	R205
Sparse_Right_X_Add	function	R206
Sparse_Right_XY_Subtract	function	R207
Set_to_Zero_Vector	function	N/A

3.3.6.1.9.1.2 LOCAL ENTITIES DESIGN

Subprograms:

The following table describes the subprograms maintained local to this part:

Name	Type	Description
RSOS	function	Instantiated version of General_Purpose_Math.General_Operations.Root_Sum_Of_Squares; required by Length function

3.3.6.1.9.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

The following generic parameters were previously described when this part was specified:

Data types:

The following table describes the generic formal types required by this part:

Name	Type	Description
Axes	scalar type	Used to dimension the exported vector type
Elements	floating point type	Data type of elements in exported vector type
Elements_Squared	floating point type	Data type resulting from multiplying two objects of type Elements

Subprograms:

The following table describes the generic formal subroutines required by this part:

Name	Type	Description
"*"	function	Multiplication operator defining the operation: Elements * Elements := Elements_Squared
Sqrt	function	Square root operator

3.3.6.1.9.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
X	Axes	Axes'FIRST	Constant used to index first element in vector
Y	Axes	Axes'SUCC(x)	Constant used to index second element in vector
Z	Axes	Axes'LAST	Constant used to index last element in vector
Zero_Vector	Vectors	0.0	Contant vector whose elements have all been set to 0.0

3.3.6.1.9.1.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.1.6 PROCESSING

The following describes the processing performed by this part:

```
   package body Vector_Operations is

--    ----------------------
--    --declaration section-
--    ----------------------

      X           : constant Axes := Axes'FIRST;
      y           : constant Axes := Axes'SUCC(X);
      z           : constant Axes := Axes'LAST;

      Zero_Vector : constant Vectors := (others => 0.0);

--    ------------------
--    --local functions-
--    ------------------

      function RSOS is new
                General_Purpose_Math.Root_Sum_of_Squares
                   (Real_Type    => Elements,
```

```
                    Squared_Type => Elements_Squared,
                    "*"          => "*",
                    Sqrt         => Sqrt);

   end Vector_Operations;
```

3.3.6.1.9.1.7 UTILIZATION OF OTHER ELEMENTS

The following library units were with'd by the Coordinate_Vector_Matrix_Algebra TLCSC:

1. General_Purpose_Math (GPMath)

UTILIZATION OF EXTERNAL ELEMENTS:

Subprograms and task entries:

The following table summarizes the external subroutines and task entries required by this part:

Name	Type	Source	Description
Root_Sum_Of_Squares	generic function	GPMath	Performs calculations necessary to compute the length of a vector

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Data types:

The following data types were previously defined in the specification of this part:

Name	Type	Range	Description
Vectors	array	N/A	One-dimensional array of Elements

3.3.6.1.9.1.8 LIMITATIONS

None.

3.3.6.1.9.1.9 LLCSC DESIGN

None.

3.3.6.1.9.1.10 UNIT DESIGN

3.3.6.1.9.1.10.1 "+" UNIT DESIGN (CATALOG #P693-0)

This unit, which is a function, calculates the result of adding two vectors. Each vector has components in the x-, y-, and z-axes of the Cartesian coordinate system.

3.3.6.1.9.1.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R050.

3.3.6.1.9.1.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.1.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Vectors	In	First vector to be added
Right	Vectors	In	Second vector to be added

3.3.6.1.9.1.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Vectors	Result of adding two input vectors

3.3.6.1.9.1.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.1.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
      function "+" (Left  : Vectors;
                    Right : Vectors) return Vectors is
```

```
--       ----------------------
--       --declaration section-
--       ----------------------

         Answer : Vectors;

--     ----------------
--     --function body-
--     ----------------

       begin

          Answer(X) := Left(X) + Right(X);
          Answer(Y) := Left(Y) + Right(Y);
          Answer(Z) := Left(Z) + Right(Z);

          return Answer;

       end "+";
```

3.3.6.1.9.1.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the top level component:

Data types:

The following table summarizes the types required by this part and defined as generic formal parameters to the Coordinate_Vector_Matrix_Algebra.Vector_-Operations LLCSC:

Name	Type	Description
Axes Elements	scalar type | floating point type	Used to dimension the exported vector type | Data type of elements in exported vector type

The following table summarizes the types required by this part and defined in the package specification of the Coordinate_Vector_Matrix_Algebra.Vector_-Operations LLCSC:

Name	Type	Range	Description
Vectors	array	N/A	One-dimensional array of Elements

Data objects:

The following table summarizes the objects required by this part and defined in the package body of the Coordinate_Vector_Matrix_Algebra.Vector_Operations LLCSC:

Name	Type	Value	Description
X	Axes	Axes'FIRST	Constant used to index first element in vector
Y	Axes	Axes'SUCC(x)	Constant used to index second element in vector
Z	Axes	Axes'LAST	Constant used to index last element in vector

3.3.6.1.9.1.10.1.8 LIMITATIONS

None.

3.3.6.1.9.1.10.2 "-" UNIT DESIGN (CATALOG #P694-0)

This unit, which is a function, calculates the result of subtracting two vectors. Each vector has components in the x-, y-, and z-axes of the Cartesian coordinate system.

3.3.6.1.9.1.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R051.

3.3.6.1.9.1.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.1.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Vectors	In	Vector to be subtracted from
Right	Vectors	In	Vector to be used as the subtrahend

3.3.6.1.9.1.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Vectors	Result of subtracting two input vectors

3.3.6.1.9.1.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.1.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
      function "-" (Left  : Vectors;
                    Right : Vectors) return Vectors is

--       ----------------------
--       --declaration section-
--       ----- ---------------

         Answer : Vectors;

--    ----------------
--    --function body-
--    ----------------

      begin

         Answer(X) := Left(X) - Right(X);
         Answer(Y) := Left(Y) - Right(Y);
         Answer(Z) := Left(Z) - Right(Z);

         return Answer;

      end "-";
```

3.3.6.1.9.1.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the top level component:

Data types:

The following table summarizes the types required by this part and defined as generic formal parameters to the Coordinate_Vector_Matrix_Algebra.Vector_-Operations LLCSC:

Name	Type	Description
Axes	scalar type	Used to dimension the exported vector type
Elements	floating point type	Data type of elements in exported vector type

The following table summarizes the types required by this part and defined in the package specification of the Coordinate_Vector_Matrix_Algebra.Vector_Operations LLCSC:

Name	Type	Range	Description
Vectors	array	N/A	One-dimensional array of Elements

Data objects:

The following table summarizes the objects required by this part and defined in the package body of the Coordinate_Vector_Matrix_Algebra.Vector_Operations LLCSC:

Name	Type	Value	Description
X	Axes	Axes'FIRST	Constant used to index first element in vector
Y	Axes	Axes'SUCC(x)	Constant used to index second element in vector
Z	Axes	Axes'LAST	Constant used to index last element in vector

3.3.6.1.9.1.10.2.8 LIMITATIONS

None.

3.3.6.1.9.1.10.3 VECTOR_LENGTH UNIT DESIGN (CATALOG #P55-0)

This unit, which is a function, calculates the length of a coordinate vector. The vector has components in the x-, y-, and z-axes of a Cartesian coordinate system.

3.3.6.1.9.1.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R208.

3.3.6.1.9.1.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.1.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Vector	Vectors	In	Vector for which a length is to be calculated

3.3.6.1.9.1.10.3.4 LOCAL DATA

None.

3.3.6.1.9.1.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.1.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
function Vector_Length (Vector : Vectors) return Elements is
begin
   return RSOS(Vector(X), Vector(Y), Vector(Z));
end Vector_Length;
```

3.3.6.1.9.1.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined in the package body of the Coordinate_Vector_Matrix_-Algebra.Vector_Operations LLCSC:

Name	Type	Description
RSOS	function	Instantiated version of General_Purpose_Math.General_ Operations.Root_Sum_Of_Squares

Data types:

The following table summarizes the types required by this part and defined as generic formal parameters to the Coordinate_Vector_Matrix_Algebra.Vector_-Operations LLCSC:

Name	Type	Description
Axes	scalar type	Used to dimension the exported vector type
Elements	floating point type	Data type of elements in exported vector type

The following table summarizes the types required by this part and defined in the package specification of the Coordinate_Vector_Matrix_Algebra.Vector_-Operations LLCSC:

Name	Type	Range	Description
Vectors	array	N/A	One-dimensional array of Elements

Data objects:

The following table summarizes the objects required by this part and defined in the package body of the Coordinate_Vector_Matrix_Algebra.Vector_Operations LLCSC:

Name	Type	Value	Description
X	Axes	Axes'FIRST	Constant used to index first element in vector
Y	Axes	Axes'SUCC(x)	Constant used to index second element in vector
Z	Axes	Axes'LAST	Constant used to index last element in vector

3.3.6.1.9.1.10.3.8 LIMITATIONS

None.

3.3.6.1.9.1.10.4 DOT_PRODUCT UNIT DESIGN (CATALOG #P56-0)

This unit, which is a function, calculates the dot product of two vectors. Each vector has components in the x-, y-, and z-axes of a Cartesian coordinate system.

3.3.6.1.9.1.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R052.

3.3.6.1.9.1.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.1.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Vector1	Vectors	In	First vector to be used for the dot product operation
Vector2	Vectors	In	Second vector to be used for the dot product operation

3.3.6.1.9.1.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Elements	Result of performing a dot product operation

3.3.6.1.9.1.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.1.10.4.6 PROCESSING

The following describes the processing performed by this part:

```
function Dot_Product (Vector1 : Vectors;
                      Vector2 : Vectors) return Elements_Squared is
```

```
--       ----------------------
--       --declaration section-
--       ----------------------

         Answer : Elements_Squared;

--     ---------------
--     --function body-
--     ---------------

       begin

          Answer := Vector1(X) * Vector2(X) +
                    Vector1(Y) * Vector2(Y) +
                    Vector1(Z) * Vector2(Z);

         return Answer;

       end Dot_Product;
```

3.3.6.1.9.1.10.4.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the top level component:

Data types:

The following table summarizes the types required by this part and defined as generic formal parameters to the Coordinate_Vector_Matrix_Algebra.Vector_-Operations LLCSC:

Name	Type	Description
Axes	scalar type	Used to dimension the exported vector type
Elements	floating point type	Data type of elements in exported vector type
Elements_Squared	floating point type	Data type resulting from multiplying two objects of type Elements

The following table summarizes the types required by this part and defined in the package specification of the Coordinate_Vector_Matrix_Algebra.Vector_-Operations LLCSC:

Name	Type	Range	Description
Vectors	array	N/A	One-dimensional array of Elements

Data objects:

The following table summarizes the objects required by this part and defined in the package body of the Coordinate_Vector_Matrix_Algebra.Vector_Operations LLCSC:

Name	Type	Value	Description
X	Axes	Axes'FIRST	Constant used to index first element in vector
Y	Axes	Axes'SUCC(x)	Constant used to index second element in vector
Z	Axes	Axes'LAST	Constant used to index last element in vector

3.3.6.1.9.1.10.4.8 LIMITATIONS

None.

3.3.6.1.9.1.10.5 SPARSE RIGHT Z ADD UNIT DESIGN (CATALOG #P57-0)

This unit, which is a function, calculates the addition of two vectors. Each vector has components in the x-, y-, and z-axes of a Cartesian coordinate system. The z-component of the second vector equals 0.

3.3.6.1.9.1.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R205.

3.3.6.1.9.1.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.1.10.5.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Vectors	In	First vector to be added
Right	Vector	In	Second vector to be added; z-component equals 0

3.3.6.1.9.1.10.5.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Vectors	Result of adding two input vectors

3.3.6.1.9.1.10.5.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.1.10.5.6 PROCESSING

The following describes the processing performed by this part:

```
      function Sparse_Right_Z_Add (Left  : Vectors;
                                   Right : Vectors) return Vectors is

--       ----------------------
--       --declaration section-
--       ----------------------

         Answer : Vectors;

--     ---------------
--     --function body-
--     ---------------

      begin

         Answer(X) := Left(X) + Right(X);
         Answer(Y) := Left(Y) + Right(Y);
         Answer(Z) := Left(Z);

         return Answer;

      end Sparse_Right_Z_Add;
```

3.3.6.1.9.1.10.5.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the top level component:

Data types:

The following table summarizes the types required by this part and defined as generic formal parameters to the Coordinate_Vector_Matrix_Algebra.Vector_-Operations LLCSC:

Name	Type	Description
Axes	scalar type	Used to dimension the exported vector type
Elements	floating point type	Data type of elements in exported vector type

The following table summarizes the types required by this part and defined in the package specification of the Coordinate_Vector_Matrix_Algebra.Vector_-Operations LLCSC:

Name	Type	Range	Description
Vectors	array	N/A	One-dimensional array of Elements

Data objects:

The following table summarizes the objects required by this part and defined in the package body of the Coordinate_Vector_Matrix_Algebra.Vector_Operations LLCSC:

Name	Type	Value	Description
X	Axes	Axes'FIRST	Constant used to index first element in vector
Y	Axes	Axes'SUCC(x)	Constant used to index second element in vector
Z	Axes	Axes'LAST	Constant used to index last element in vector

3.3.6.1.9.1.10.5.8 LIMITATIONS

None.

3.3.6.1.9.1.10.6 SPARSE RIGHT X ADD UNIT DESIGN (CATALOG #P58-0)

This unit, which is a function, calculates the result of adding two vectors. Each vector has components in the x-, y-, and z-axes of the Cartesian coordinate system. The x-component of the second vector equals 0.

3.3.6.1.9.1.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R206.

3.3.6.1.9.1.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.1.10.6.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Vectors	In	First vector to be added
Right	Vectors	In	Second vector to be added; x-component equals 0

3.3.6.1.9.1.10.6.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Vectors	Result of adding two input vectors

3.3.6.1.9.1.10.6.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.1.10.6.6 PROCESSING

The following describes the processing performed by this part:

```
      function Sparse_Right_X_Add (Left  : Vectors;
                                   Right : Vectors) return Vectors is

--       ----------------------
--       --declaration section-
--       ----------------------

         Answer : Vectors;

--     ---------------
--     --function body-
--     ---------------

      begin
```

```
        Answer(X) := Left(X);
        Answer(Y) := Left(Y) + Right(Y);
        Answer(Z) := Left(Z) + Right(Z);

        return Answer;

    end Sparse_Right_X_Add;
```

3.3.6.1.9.1.10.6.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the top level component:

Data types:

The following table summarizes the types required by this part and defined as generic formal parameters to the Coordinate_Vector_Matrix_Algebra.Vector_-Operations LLCSC:

Name	Type	Description
Axes Elements	scalar type floating point type	Used to dimension the exported vector type Data type of elements in exported vector type

The following table summarizes the types required by this part and defined in the package specification of the Coordinate_Vector_Matrix_Algebra.Vector_-Operations LLCSC:

Name	Type	Range	Description
Vectors	array	N/A	One-dimensional array of Elements

Data objects:

The following table summarizes the objects required by this part and defined in the package body of the Coordinate_Vector_Matrix_Algebra.Vector_Operations LLCSC:

Name	Type	Value	Description
X	Axes	Axes'FIRST	Constant used to index first element in vector
Y	Axes	Axes'SUCC(x)	Constant used to index second element in vector
Z	Axes	Axes'LAST	Constant used to index last element in vector

3.3.6.1.9.1.10.6.8 LIMITATIONS

None.

3.3.6.1.9.1.10.7 SPARSE RIGHT XY SUBTRACT UNIT DESIGN (CATALOG #P59-0)

This unit, which is a function, calculates the result of subtracting two vectors. Each vector has components in the x-, y-, and z-axes of the Cartesian coordinate system. The x- and y-components of the second vector equal 0.

3.3.6.1.9.1.10.7.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R207.

3.3.6.1.9.1.10.7.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.1.10.7.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Vectors	In	First vector to be used in subtraction
Right	Vectors	In	Second vector to be used in subtraction; x- and y- components equal 0

3.3.6.1.9.1.10.7.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Vectors	Result of subtracting two input vectors

3.3.6.1.9.1.10.7.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.1.10.7.6 PROCESSING

The following describes the processing performed by this part:

```
      function Sparse_Right_XY_Subtract (Left  : Vectors;
                                         Right : Vectors) return Vectors is

--       ---------------------
--       --declaration section-
--       ---------------------

         Answer : Vectors;

--     ---------------
--     --function body-
--     ---------------

      begin

         Answer(X) := Left(X);
         Answer(Y) := Left(Y);
         Answer(Z) := Left(Z) - Right(Z);

         return Answer;

      end Sparse_Right_XY_Subtract;
```

3.3.6.1.9.1.10.7.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the top level component:

Data types:

The following table summarizes the types required by this part and defined as generic formal parameters to the Coordinate_Vector_Matrix_Algebra.Vector_-Operations LLCSC:

Name	Type	Description
Axes Elements	scalar type | floating point type	Used to dimension the exported vector type| Data type of elements in exported vector type

The following table summarizes the types required by this part and defined in the package specification of the Coordinate_Vector_Matrix_Algebra.Vector_-Operations LLCSC:

Name	Type	Range	Description
Vectors	array	N/A	One-dimensional array of Elements

Data objects:

The following table summarizes the objects required by this part and defined in the package body of the Coordinate_Vector_Matrix_Algebra.Vector_Operations LLCSC:

Name	Type	Value	Description
X	Axes	Axes'FIRST	Constant used to index first element in vector
Y	Axes	Axes'SUCC(x)	Constant used to index second element in vector
Z	Axes	Axes'LAST	Constant used to index last element in vector

3.3.6.1.9.1.10.7.8 LIMITATIONS

None.

3.3.6.1.9.1.10.8 SET_TO_ZERO_VECTOR UNIT DESIGN (CATALOG #P60-0)

This function returns a vector whose elements have all been set to 0.0.

3.3.6.1.9.1.10.8.1 REQUIREMENTS ALLOCATION

N/A

3.3.6.1.9.1.10.8.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.1.10.8.3 INPUT/OUTPUT

None.

3.3.6.1.9.1.10.8.4 LOCAL DATA

None.

3.3.6.1.9.1.10.8.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.1.10.8.6 PROCESSING

The following describes the processing performed by this part:

```
function Set_to_Zero_Vector return Vectors is

begin

   return Zero_Vector;

end Set_to_Zero_Vector;
```

3.3.6.1.9.1.10.8.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the top level component:

Data types:

The following table summarizes the types required by this part and defined as generic formal parameters to the Coordinate_Vector_Matrix_Algebra.Vector_-Operations LLCSC:

Name	Type	Description
Axes	scalar type	Used to dimension the exported vector type
Elements	floating point type	Data type of elements in exported vector type

The following table summarizes the types required by this part and defined in the package specification of the Coordinate_Vector_Matrix_Algebra.Vector_-Operations LLCSC:

Name	Type	Range	Description
Vectors	array	N/A	One-dimensional array of Elements

Data objects:

The following table summarizes the objects required by this part and defined in the package body of the Coordinate_Vector_Matrix_Algebra.Vector_Operations LLCSC:

Name	Type	Value	Description
Zero_Vector	Vectors	0.0	Contant vector whose elements have all been set to 0.0

3.3.6.1.9.1.10.8.8 LIMITATIONS

None.

3.3.6.1.9.2 VECTOR_SCALAR_OPERATIONS (BODY) PACKAGE DESIGN (CATALOG #P64-0)

This LLCSC provides the functions to allow multiplication or division of each element of a vector by a scalar.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.1.9.2.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of requirements to this part:

Name	Requirement Allocation
"*"	R054
Sparse_X_Vector_Scalar_Multiply	R209
"/"	R055

3.3.6.1.9.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.2.3 INPUT/OUTPUT

None.

3.3.6.1.9.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
x	constant Axes	Used to index first element in vector
y	constant Axes	Used to index second element in vector
z	constant Axes	Used to index last element in vector

3.3.6.1.9.2.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.2.6 PROCESSING

The following describes the processing performed by this part:

```
   package body Vector_Scalar_Operations is

--    ----------------------
--    --declaration section-
--    ----------------------

      x : constant Axes := Axes'FIRST;
      y : constant Axes := Axes'SUCC(x);
      z : constant Axes := Axes'LAST;

   end Vector_Scalar_Operations;
```

3.3.6.1.9.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.1.9.2.8 LIMITATIONS

None.

3.3.6.1.9.2.9 LLCSC DESIGN

None.

3.3.6.1.9.2.10 UNIT DESIGN

3.3.6.1.9.2.10.1 "*" UNIT DESIGN (CATALOG #P700-0)

This unit, which is a function, multiples each element of a vector by a scalar. The vector has 3 elements which are components in the x-, y-, and z-axes of a Cartesian coordinate system.

3.3.6.1.9.2.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R054.

3.3.6.1.9.2.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.2.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

Name	Type	Mode	Description
Vector	Vectors1	In	Vector to be scaled
Multiplier	Scalars	In	Value to be used when multiplying the elements of the vector

3.3.6.1.9.2.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Vectors2	Scaled vector as calculated by this part

3.3.6.1.9.2.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.2.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
      function "*" (Vector     : Vectors1;
                    Multiplier : Scalars) return Vectors2 is

--       ----------------------
--       --declaration section-
--       ----------------------

         Answer : Vectors2;

--    --------------------
--    --begin function "*"
--    --------------------
```

```
   begin

      Answer(x) := Vector(x) * Multiplier;
      Answer(y) := Vector(y) * Multiplier;
      Answer(z) := Vector(z) * Multiplier;

      return Answer;

   end "*";
```

3.3.6.1.9.2.10.1.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined as generic parameters to the Coordinate_Vector_Matrix_-Algebra.Vector_Scalar_Operations LLCSC:

Name	Type	Description
"*"	function	Multiplication operator used to define the operation: Elements1 * Scalars := Elements2

Data types:

The following table summarizes the types required by this part and defined as generic parameters to the Coordinate_Vector_Matrix_Algebra.Vector_Scalar_-Operations LLCSC:

Name	Type	Description
Axes	scalar type	Used to dimension imported vector types
Elements1	floating point type	Type of elements on Vectors1
Elements2	floating point type	Type of elements on Vectors2
Scalars	floating point type	Data type of scale factors

3.3.6.1.9.2.10.1.8 LIMITATIONS

None.

3.3.6.1.9.2.10.2 "/" UNIT DESIGN (CATALOG #P699-0)

This unit, which is a function, provides the capability to divide each element of a vector by a scalar. The vector has three elements which are components in the x-, y-, and z-axes of a Cartesian coordinate system.

3.3.6.1.9.2.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R055.

3.3.6.1.9.2.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.2.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Vector	Vector2	In	Vector to be scaled
Divisor	Scalars		Value each element of the vector is to be divided by

3.3.6.1.9.2.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Vectors1	Scaled vector as calculated by this part

3.3.6.1.9.2.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.2.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
function "/" (Vector  : Vectors2;
              Divisor : Scalars) return Vectors1 is
```

```
--       ---------------------
--       --declaration section-
--       ----------------------

         Answer : Vectors1;

--     --------------------
--     --begin function "/"
--     --------------------

       begin

          Answer(x) := Vector(x) / Divisor;
          Answer(y) := Vector(y) / Divisor;
          Answer(z) := Vector(z) / Divisor;

          return Answer;

       end "/";
```

3.3.6.1.9.2.10.2.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined as generic parameters to the Coordinate_Vector_Matrix_-Algebra.Vector_Scalar_Operations LLCSC:

Name	Type	Description
"/"	function	Division operator used to define the operation: Elements2 / Scalars := Elements1

Data types:

The following table summarizes the types required by this part and defined as generic parameters to the Coordinate_Vector_Matrix_Algebra.Vector_Scalar_-Operations LLCSC:

Name	Type	Description
Axes	scalar type	Used to dimension imported vector types
Elements1	floating point type	Type of elements on Vectors1
Elements2	floating point type	Type of elements on Vectors2
Scalars	floating point type	Data type of scale factors

3.3.6.1.9.2.10.2.8 LIMITATIONS

None.

3.3.6.1.9.2.10.3 SPARSE X VECTOR SCALAR MULTIPLY UNIT DESIGN (CATALOG #P65-0)

This unit, which is a function, provides the capability to multiply each element of a vector by a scalar. The vector has 3 elements which are components in the x-, y-, and z-axes of a Cartesian coordinate system. The x-component of the vector equals 0.

3.3.6.1.9.2.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R209.

3.3.6.1.9.2.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.2.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Vector	Vectors1	In	Vector to be scaled; x-component equals 0
Multiplier	Scalars	In	Value each vector element is multiplied by

3.3.6.1.9.2.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Vectors2	Scaled vector as calculated by this part

3.3.6.1.9.2.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.2.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
      function Sparse_X_Vector_Scalar_Multiply
                  (Vector     : Vectors1;
                   Multiplier : Scalars) return Vectors2 is

--       ---------------------
--       --declaration section-
--       ---------------------

         Answer : Vectors2;

--    ------------------------------------------------
--    --begin function Sparse_X_Vector_Scalar_Multiply
--    ------------------------------------------------

      begin

         Answer(X) := 0.0;
         Answer(Y) := Vector(Y) * Multiplier;
         Answer(Z) := Vector(Z) * Multiplier;

         return Answer;

      end Sparse_X_Vector_Scalar_Multiply;
```

3.3.6.1.9.2.10.3.7 UTILIZATION OF OTHER ELEMENTS

UTILIZATION OF OTHER ELEMENTS IN TOP LEVEL COMPONENT:

The following tables describe the elements used by this part but defined elsewhere in the parent top level component:

Subprograms and task entries:

The following table summarizes the subroutines and task entries required by this part and defined as generic parameters to the Coordinate_Vector_Matrix_-Algebra.Vector_Scalar_Operations LLCSC:

Name	Type	Description
"*"	function	Multiplication operator used to define the operation: Elements1 * Scalars := Elements2

Data types:

The following table summarizes the types required by this part and defined as generic parameters to the Coordinate_Vector_Matrix_Algebra.Vector_Scalar_-Operations LLCSC:

Name	Type	Description
Axes	scalar type	Used to dimension imported vector types
Elements1	floating point type	Type of elements on Vectors1
Elements2	floating point type	Type of elements on Vectors2
Scalars	floating point type	Data type of scale factors

3.3.6.1.9.2.10.3.8 LIMITATIONS

None.

3.3.6.1.9.3 MATRIX OPERATIONS PACKAGE DESIGN (CATALOG #P61-0)

This package provides general operations on a two-dimensional coordinate matrix.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.1.9.3.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of requirements to this part's units:

Name	Type	Requirement Allocation
"+" (matrices + matrices)	function	R070
"-" (matrices - matrices)	function	R071
"+" (matrices + elements)	function	R060
"-" (matrices - elements)	function	R067
Set_To_Identity_Matrix	function	R072
Set_To_Zero_Matrix	function	R078

3.3.6.1.9.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.3.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Previously described in package specification.

3.3.6.1.9.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
x	Axes	Axes'FIRST	Constant used to index first element in matrix row and/or column
y	Axes	Axes'SUCC(X)	Constant used to index second element in matrix row and/or column
z	Axes	Axes'LAST	Constant used to index last element in matrix row and/or column
Identity_ Matrix	Matrices		Identity matrix with diagonal elements set to 1.0 and all other elements set to 0.0
Zero_ Matrix	Matrices	0.0	Constant zero matrix with all elements set to 0.0

3.3.6.1.9.3.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.3.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Coordinate_Vector_Matrix_Algebra)
package body Matrix_Operations is

-- ----------------------
-- --declaration section-
-- ----------------------

   X                : constant Axes      := Axes'FIRST;
   Y                : constant Axes      := Axes'SUCC(X);
   Z                : constant Axes      := Axes'LAST;

-- --the diagonal elements of Identity_Matrix will be set to 1.0 during
-- --package initialization
```

```
   Identity_Matrix : Matrices := (others => (others => 0.0));

   Zero_Matrix     : constant Matrices := (others => (others => 0.0));

-- --------------------
-- --subroutine bodies-
-- --------------------

   function Set_to_Identity_Matrix return Matrices is separate;

   function Set_to_Zero_Matrix return Matrices is separate;


---------------------------------
--begin package Matrix_Operations-
---------------------------------

begin

-- --initialize diagonal elements of Identity_Matrix, remaining elements have
-- --already been set to 0.0

   Identity_Matrix(X,X) := 1.0;
   Identity_Matrix(Y,Y) := 1.0;
   Identity_Matrix(Z,Z) := 1.0;

end Matrix_Operations;
```

3.3.6.1.9.3.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.1.9.3.8 LIMITATIONS

None.

3.3.6.1.9.3.9 LLCSC DESIGN

None.

3.3.6.1.9.3.10 UNIT DESIGN

3.3.6.1.9.3.10.1 "+" (MATRICES + MATRICES) UNIT DESIGN (CATALOG #P695-0)

This unit, which is a function, provides the capability to calculated the result of adding two matrices. Each matrix is a 3 x 3 matrix having 9 elements which are components in the x-, y-, and z-axes of a Cartesian coordinate system.

3.3.6.1.9.3.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R070.

3.3.6.1.9.3.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.3.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Matrices	In	First matrix to be added
Right	Matrices	In	Second vector to be added

3.3.6.1.9.3.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Matrices	Result of adding two input matrices

3.3.6.1.9.3.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.3.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   function "+" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Matrices;

-- ----------------------------------------
-- --begin function "+" (matrices + matrices)
```

```
-- ----------------------------------------

   begin

      Answer(X,X) := Left(X.X) + Right(X,X);
      Answer(X,Y) := Left(X,Y) + Right(X,Y);
      Answer(X,Z) := Left(X,Z) + Right(X,Z);

      Answer(Y,X) := Left(Y,X) + Right(Y,X);
      Answer(Y,Y) := Left(Y,Y) + Right(Y,Y);
      Answer(Y,Z) := Left(Y,Z) + Right(Y,Z);

      Answer(Z,X) := Left(Z,X) + Right(Z,X);
      Answer(Z,Y) := Left(Z,Y) + Right(Z,Y);
      Answer(Z,Z) := Left(Z,Z) + Right(Z,Z);

      return Answer;

   end "+";
```

3.3.6.1.9.3.10.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.1.9.3.10.1.8 LIMITATIONS

None.

3.3.6.1.9.3.10.2 "-" (MATRICES - MATRICES) UNIT DESIGN (CATALOG #P696-0)

This unit, which is a function, provides the capability to calculate the result of subtraction two matrices. Each matrix is a 3 x 3 matrix having 9 elements which are components in the x-, y-, and z-axes of a Cartesian coordinate system.

3.3.6.1.9.3.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R071.

3.3.6.1.9.3.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.3.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Matrices	In	First matrix to be treated as the minuend
Right	Matrices	In	Second matrix to be treated as the subtrahend

3.3.6.1.9.3.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Matrices	Result of subtracting two input vectors

3.3.6.1.9.3.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.3.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
   function "-" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Matrices;

-- ----------------------------------------
-- --begin function "-" (matrices - matrices)
-- ----------------------------------------

   begin

      Answer(X,X) := Left(X,X) - Right(X,X);
      Answer(X,Y) := Left(X,Y) - Right(X,Y);
      Answer(X,Z) := Left(X,Z) - Right(X,Z);

      Answer(Y,X) := Left(Y,X) - Right(Y,X);
      Answer(Y,Y) := Left(Y,Y) - Right(Y,Y);
      Answer(Y,Z) := Left(Y,Z) - Right(Y,Z);

      Answer(Z,X) := Left(Z,X) - Right(Z,X);
      Answer(Z,Y) := Left(Z,Y) - Right(Z,Y);
      Answer(Z,Z) := Left(Z,Z) - Right(Z,Z);
```

```
      return Answer;

   end "-";
```

3.3.6.1.9.3.10.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.1.9.3.10.2.8 LIMITATIONS

None.

3.3.6.1.9.3.10.3 "+" (MATRICES + ELEMENTS) UNIT DESIGN (CATALOG #P697-0)

This unit, which is a function, provides the capability to add a scalar value to each element of a matrix. The matrix is a 3 x 3 matrix having 9 elements which are components in the x-, y-, and z-axes of a Cartesian coordinate system.

3.3.6.1.9.3.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R060.

3.3.6.1.9.3.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.3.10.3.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	In	Matrix to be scaled
Addend	Elements	In	Value to be added to each element of the matrix

3.3.6.1.9.3.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Matrices	Scaled matrix

3.3.6.1.9.3.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.3.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
   function "+" (Matrix : Matrices;
                 Addend : Elements) return Matrices is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Matrices;

-- ------------------------------------------
-- --begin function "+" (matrices + elements)-
-- ------------------------------------------

   begin

      Answer(X,X) := Matrix(X,X) + Addend;
      Answer(X,Y) := Matrix(X,Y) + Addend;
      Answer(X,Z) := Matrix(X,Z) + Addend;

      Answer(Y,X) := Matrix(Y,X) + Addend;
      Answer(Y,Y) := Matrix(Y,Y) + Addend;
      Answer(Y,Z) := Matrix(Y,Z) + Addend;

      Answer(Z,X) := Matrix(Z,X) + Addend;
      Answer(Z,Y) := Matrix(Z,Y) + Addend;
      Answer(Z,Z) := Matrix(Z,Z) + Addend;

      return Answer;

   end "+";
```

3.3.6.1.9.3.10.3.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.1.9.3.10.3.8 LIMITATIONS

None.

3.3.6.1.9.3.10.4 "-" (MATRICES - ELEMENTS) UNIT DESIGN (CATALOG #P698-0)

This unit, which is a function, provides the capability to subtract a scalar value from each element of a matrix. The matrix is a 3 x 3 matrix having 9 elements which are components in the x-, y-, and z-axes of a Cartesian coordinate system.

3.3.6.1.9.3.10.4.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R067.

3.3.6.1.9.3.10.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.3.10.4.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	In	Matrix to be scaled
Subtrahend	Elements	In	Value to be subtracted from each element in the matrix

3.3.6.1.9.3.10.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Matrices	Scaled matrix as calculated by this part

3.3.6.1.9.3.10.4.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.3.10.4.6 PROCESSING

The following describes the processing performed by this part:

```
   function "-" (Matrix      : Matrices;
                 Subtrahend : Elements) return Matrices is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Matrices;

-- ----------------
-- --function body-
-- ----------------

   begin

      Answer(X,X) := Matrix(X,X) - Subtrahend;
      Answer(X,Y) := Matrix(X,Y) - Subtrahend;
      Answer(X,Z) := Matrix(X,Z) - Subtrahend;

      Answer(Y,X) := Matrix(Y,X) - Subtrahend;
      Answer(Y,Y) := Matrix(Y,Y) - Subtrahend;
      Answer(Y,Z) := Matrix(Y,Z) - Subtrahend;

      Answer(Z,X) := Matrix(Z,X) - Subtrahend;
      Answer(Z,Y) := Matrix(Z,Y) - Subtrahend;
      Answer(Z,Z) := Matrix(Z,Z) - Subtrahend;

      return Answer;

   end "-";
```

3.3.6.1.9.3.10.4.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.1.9.3.10.4.8 LIMITATIONS

None.

3.3.6.1.9.3.10.5 SET TO IDENTITY MATRIX UNIT DESIGN (CATALOG #P62-0)

This part, which is a function, provides the capability to initialize a matrix to an identity matrix. The matrix shall be a 3 x 3 matrix with 9 elements which are components of the x-, y-, and z-axes of the Cartesian coordinate system.

3.3.6.1.9.3.10.5.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R072.

3.3.6.1.9.3.10.5.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.3.10.5.3 INPUT/OUTPUT

None.

3.3.6.1.9.3.10.5.4 LOCAL DATA

None.

3.3.6.1.9.3.10.5.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.3.10.5.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Coordinate_Vector_Matrix_Algebra.Matrix_Operations)
function Set_to_Identity_Matrix return Matrices is

begin

    return Identity_Matrix;

end Set_to_Identity_Matrix;
```

3.3.6.1.9.3.10.5.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.1.9.3.10.5.8 LIMITATIONS

None.

3.3.6.1.9.3.10.6 SET TO ZERO MATRIX UNIT DESIGN (CATALOG #P63-0)

This unit, which is a function, provides the capability to initialize each element of a matrix to zero. The matrix is a 3 x 3 matrix having 9 elements which are components of the x-, y-, and z-axes of the Cartesian coordinate system.

3.3.6.1.9.3.10.6.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R078.

3.3.6.1.9.3.10.6.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.3.10.6.3 INPUT/OUTPUT

None.

3.3.6.1.9.3.10.6.4 LOCAL DATA

None.

3.3.6.1.9.3.10.6.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.3.10.6.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Coordinate_Vector_Matrix_Algebra.Matrix_Operations)
function Set_to_Zero_Matrix return Matrices is

begin

   return Zero_Matrix;

end Set_to_Zero_Matrix;
```

3.3.6.1.9.3.10.6.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.1.9.3.10.6.8 LIMITATIONS

None.

3.3.6.1.9.4 MATRIX SCALAR OPERATIONS PACKAGE DESIGN (CATALOG #P66-0)

This LLCSC, which is a package body, provides the functions to allow multiplication or division of each element of a matrix by a scalar.

The decomposition for this part is the same as that shown in the Top-Level Design Document.

3.3.6.1.9.4.1 REQUIREMENTS ALLOCATION

The following chart summarizes the allocation of requirements to this part:

Name	Requirement Allocation
"*"	R056
"/"	R057

3.3.6.1.9.4.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.4.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Previously described in package specification.

3.3.6.1.9.4.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
x	constant Axes	Used to index first element in matrix row and/or column
y	constant Axes	Used to index second element in matrix row and/or column
z	constant Axes	Used to index last element in matrix row and/or column

3.3.6.1.9.4.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.4.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Coordinate_Vector_Matrix_Algebra)
package body Matrix_Scalar_Operations is

-- ----------------------
-- --declaration section-
-- ----------------------
```

```
   X : constant Axes := Axes'FIRST;
   Y : constant Axes := Axes'SUCC(X);
   Z : constant Axes := Axes'LAST;

end Matrix_Scalar_Operations;
```

3.3.6.1.9.4.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.1.9.4.8 LIMITATIONS

None.

3.3.6.1.9.4.9 LLCSC DESIGN

None.

3.3.6.1.9.4.10 UNIT DESIGN

3.3.6.1.9.4.10.1 "*" UNIT DESIGN (CATALOG #P701-0)

This unit, which is a function, provides the capability of multiplying each element of a matrix by a scalar. The matrix is a 3 x 3 matrix having 9 elements which are components in the x-, y-, and z-axes of a Cartesian coordinate system.

3.3.6.1.9.4.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R056.

3.3.6.1.9.4.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.4.10.1.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices1	In	Matrix to be scaled
Multiplier	Scalars	In	Value used to multiply each element of the matrix

3.3.6.1.9.4.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Value	Description
Answer	Matrices2	N/A	Scaled matrix as calculated by this part

3.3.6.1.9.4.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.4.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
   function "*" (Matrix     : Matrices1;
                 Multiplier : Scalars) return Matrices2 is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Matrices2;

-- ---------------------
-- --begin function "*"-
-- ---------------------

   begin

      Answer(X,X) := Matrix(X,X) * Multiplier;
      Answer(X,Y) := Matrix(X,Y) * Multiplier;
      Answer(X,Z) := Matrix(X,Z) * Multiplier;
      Answer(Y,X) := Matrix(Y,X) * Multiplier;
      Answer(Y,Y) := Matrix(Y,Y) * Multiplier;
      Answer(Y,Z) := Matrix(Y,Z) * Multiplier;
      Answer(Z,X) := Matrix(Z,X) * Multiplier;
      Answer(Z,Y) := Matrix(Z,Y) * Multiplier;
      Answer(Z,Z) := Matrix(Z,Z) * Multiplier;

      return Answer;

   end "*";
```

3.3.6.1.9.4.10.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.1.9.4.10.1.8 LIMITATIONS

None.

3.3.6.1.9.4.10.2 "/" UNIT DESIGN (CATALOG #P702-0)

This unit, which is a function, provides the capability of dividing each element of a matrix by a scalar. The matrix is a 3 x 3 matrix having 9 elements which are components in the x-, y-, and z-axes of a Cartesian coordinate system.

3.3.6.1.9.4.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R057.

3.3.6.1.9.4.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.9.4.10.2.3 INPUT/OUTPUT

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices2	In	Matrix to be scaled
Divisor	Scalars		Value used to divide each element of the matrix

3.3.6.1.9.4.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
Answer	Matrices1	Scaled matrix as calculated by this part

3.3.6.1.9.4.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.1.9.4.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
   function "/" (Matrix  : Matrices2;
                 Divisor : Scalars) return Matrices1 is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Matrices1;

-- ----------------
-- --function body-
-- ----------------

   begin

      Answer(X,X) := Matrix(X,X) / Divisor;
      Answer(X,Y) := Matrix(X,Y) / Divisor;
      Answer(X,Z) := Matrix(X,Z) / Divisor;

      Answer(Y,X) := Matrix(Y,X) / Divisor;
      Answer(Y,Y) := Matrix(Y,Y) / Divisor;
      Answer(Y,Z) := Matrix(Y,Z) / Divisor;

      Answer(Z,X) := Matrix(Z,X) / Divisor;
      Answer(Z,Y) := Matrix(Z,Y) / Divisor;
      Answer(Z,Z) := Matrix(Z,Z) / Divisor;

      return Answer;

   end "/";
```

3.3.6.1.9.4.10.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.1.9.4.10.2.8 LIMITATIONS

None.

3.3.6.1.10 UNIT DESIGN

3.3.6.1.10.1 CROSS PRODUCT UNIT DESIGN (CATALOG #P67-0)

This LLCSC, which is a function, provides the capability of calculating the cross product of two vectors. Each vector has three elements which are components in the x-, y-, and z-axes of a Cartesian coordinate system.

3.3.6.1.10.1.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R053.

3.3.6.1.10.1.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.10.1.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Previously described in package specification.

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Left	Left_Vectors	N/A	Matrix to be used on the left side of the cross-product operation
Right	Right_Vectors	N/A	Matrix to be used on the right side of the cross-product operation

3.3.6.1.10.1.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
x	constant Axes	Used to index first element in vector
y	constant Axes	Used to index second element in vector
z	constant Axes	Used to index last element in vector
Answer	Result_Vectors	Vector resulting from cross product operation on two input vectors

3.3.6.1.10.1.5 PROCESS CONTROL

Not applicable.

3.3.6.1.10.1.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Coordinate_Vector_Matrix_Algebra)
function Cross_Product (Left  : Left_Vectors;
                        Right : Right_Vectors) return Result_Vectors is

-- ----------------------
-- --declaration section-
-- ----------------------

   X      : constant Axes := Axes'FIRST;
   Y      : constant Axes := Axes'SUCC(X);
   Z      : constant Axes := Axes'LAST;

   Answer : Result_Vectors;

------------------------------
--begin function Cross_Product
------------------------------

begin

   Answer(X) := Left(Y) * Right(Z) - Left(Z) * Right(Y);
   Answer(Y) := Left(Z) * Right(X) - Left(X) * Right(Z);
   Answer(Z) := Left(X) * Right(Y) - Left(Y) * Right(X);

   return Answer;

end Cross_Product;
```

3.3.6.1.10.1.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.1.10.1.8 LIMITATIONS

None.

3.3.6.1.10.2 MATRIX VECTOR MULTIPLY UNIT DESIGN (CATALOG #P68-0)

This LLCSC, which is a function, provides the capability to multiply a matrix by a vector with the result being a vector. The matrix is a 3 x 3 matrix having 9 elements. The vectors have 3 elements. The elements are components in the x-, y-, and z-axes of a Cartesian coordinate system.

3.3.6.1.10.2.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R049.

3.3.6.1.10.2.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.10.2.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Previously described in package specification.

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix	Matrices	In	Matrix to be used in calculations
Vector	Input_Vectors	In	Vector to be used in calculations

3.3.6.1.10.2.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
x	constant Axes	Used to index first row/column in matrix and first element in vectors
y	constant Axes	Used to index second row/column in matrix and second element in vectors
z	constant Axes	Used to index last row/column in matrix and last element in vectors
Answer	Output_Vectors	Vector resulting from multiplication operation

3.3.6.1.10.2.5 PROCESS CONTROL

Not applicable.

3.3.6.1.10.2.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Coordinate_Vector_Matrix_Algebra)
function Matrix_Vector_Multiply
          (Matrix : Matrices;
           Vector : Input_Vectors) return Output_Vectors is
```

```
-- ---------------------
-- --declaration section-
-- ---------------------

   X : constant Axes := Axes'FIRST;
   Y : constant Axes := Axes'SUCC(X);
   Z : constant Axes := Axes'LAST;

   Answer : Output_Vectors;

--------------------------------------
--begin function Matrix_Vector_Multiply
--------------------------------------

begin

   Answer(X) := Matrix(X,X) * Vector(X) +
                Matrix(X,Y) * Vector(Y) +
                Matrix(X,Z) * Vector(Z);

   Answer(Y) := Matrix(Y,X) * Vector(X) +
                Matrix(Y,Y) * Vector(Y) +
                Matrix(Y,Z) * Vector(Z);

   Answer(Z) := Matrix(Z,X) * Vector(X) +
                Matrix(Z,Y) * Vector(Y) +
                Matrix(Z,Z) * Vector(Z);

   return Answer;

end Matrix_Vector_Multiply;
```

3.3.6.1.10.2.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.1.10.2.8 LIMITATIONS

None.

3.3.6.1.10.3 MATRIX MATRIX MULTIPLY UNIT DESIGN (CATALOG #P69-0)

This LLCSC, which is a function, provides the capability to multiply two matrices. Both matrices are a 3 x 3 matrix having 9 elements which are components in the x-, y-, and z-axes of a Cartesian coordinate system.

3.3.6.1.10.3.1 REQUIREMENTS ALLOCATION

This part meets CAMP requirement R068.

3.3.6.1.10.3.2 LOCAL ENTITIES DESIGN

None.

3.3.6.1.10.3.3 INPUT/OUTPUT

GENERIC PARAMETERS:

Previously described in package specification.

FORMAL PARAMETERS:

The following table describes this part's formal parameters:

Name	Type	Mode	Description
Matrix1	Left_Matrices	In	First matrix used for multiplication operation
Matrix2	Right_Matrices	In	Second matrix used for multiplication operation

3.3.6.1.10.3.4 LOCAL DATA

Data objects:

The following table describes the data objects maintained by this part:

Name	Type	Description
x	constant Axes	Used to index first element in matrix row and/or column
y	constant Axes	Used to index second element in matrix row and/or column
z	constant Axes	Used to index last element in matrix row and/or column
Answer	Result_Matrices	Matrix resulting from multiplying two input matrices

3.3.6.1.10.3.5 PROCESS CONTROL

Not applicable.

3.3.6.1.10.3.6 PROCESSING

The following describes the processing performed by this part:

```
separate (Coordinate_Vector_Matrix_Algebra)
function Matrix_Matrix_Multiply
          (Matrix1 : Left_Matrices;
```

```
            Matrix2 : Right_Matrices) return Result_Matrices is

-- ----------------------
-- --declaration section-
-- ----------------------

   X      : constant Axes := Axes'FIRST;
   Y      : constant Axes := Axes'SUCC(X);
   Z      : constant Axes := Axes'LAST;

   answer : Result_Matrices;

-- ----------------
-- --function body-
-- ----------------

   begin

--    --first row

      answer(X,X) := Matrix1(X,X) * Matrix2(X,X) +
                     Matrix1(X,Y) * Matrix2(Y,X) +
                     Matrix1(X,Z) * Matrix2(Z,X);

      answer(X,Y) := Matrix1(X,X) * Matrix2(X,Y) +
                     Matrix1(X,Y) * Matrix2(Y,Y) +
                     Matrix1(X,Z) * Matrix2(Z,Y);

      answer(X,Z) := Matrix1(X,X) * Matrix2(X,Z) +
                     Matrix1(X,Y) * Matrix2(Y,Z) +
                     Matrix1(X,Z) * Matrix2(Z,Z);

--    --second row

      answer(Y,X) := Matrix1(Y,X) * Matrix2(X,X) +
                     Matrix1(Y,Y) * Matrix2(Y,X) +
                     Matrix1(Y,Z) * Matrix2(Z,X);

      answer(Y,Y) := Matrix1(Y,X) * Matrix2(X,Y) +
                     Matrix1(Y,Y) * Matrix2(Y,Y) +
                     Matrix1(Y,Z) * Matrix2(Z,Y);

      answer(Y,Z) := Matrix1(Y,X) * Matrix2(X,Z) +
                     Matrix1(Y,Y) * Matrix2(Y,Z) +
                     Matrix1(Y,Z) * Matrix2(Z,Z);

--    --third row

      answer(Z,X) := Matrix1(Z,X) * Matrix2(X,X) +
                     Matrix1(Z,Y) * Matrix2(Y,X) +
                     Matrix1(Z,Z) * Matrix2(Z,X);

      answer(Z,Y) := Matrix1(Z,X) * Matrix2(X,Y) +
                     Matrix1(Z,Y) * Matrix2(Y,Y) +
                     Matrix1(Z,Z) * Matrix2(Z,Y);

      answer(Z,Z) := Matrix1(Z,X) * Matrix2(X,Z) +
```

```
                    Matrix1(Z,Y) * Matrix2(Y,Z) +
                    Matrix1(Z,Z) * Matrix2(Z,Z);

   return answer;

end Matrix_Matrix_Multiply;
```

3.3.6.1.10.3.7 UTILIZATION OF OTHER ELEMENTS

None.

3.3.6.1.10.3.8 LIMITATIONS

None.

(This page left intentionally blank.)

```
with General_Purpose_Math;
package body Coordinate_Vector_Matrix_Algebra is

   package body Matrix_Operations is separate;

   package body Matrix_Scalar_Operations is separate;

   function Cross_Product (Left  : Left_Vectors;
                           Right : Right_Vectors)
                          return Result_Vectors is separate;

   function Matrix_Vector_Multiply (Matrix : Matrices;
                                    Vector : Input_Vectors)
                                   return Output_Vectors is separate;

   function Matrix_Matrix_Multiply (Matrix1 : Left_Matrices;
                                    Matrix2 : Right_Matrices)
                                   return Result_Matrices is separate;

pragma PAGE;
   package body Vector_Operations is

--     ----------------------
--     --declaration section-
--     ----------------------

       X           : constant Axes := Axes'FIRST;
       Y           : constant Axes := Axes'SUCC(X);
       Z           : constant Axes := Axes'LAST;

       Zero_Vector : constant Vectors := (others => 0.0);

--     ------------------
--     --local functions-
--     ------------------

       function Rsos is new
                 General_Purpose_Math.Root_Sum_Of_Squares
                    (Real_Type    => Elements,
                     Squared_Type => Elements_Squared,
                     "*"          => "*",
                     Sqrt         => Sqrt);

pragma PAGE;
       function "+" (Left  : Vectors;
                     Right : Vectors) return Vectors is

--        ----------------------
--        --declaration section-
--        ----------------------

          Answer : Vectors;

--     ---------------
--     --function body-
--     ---------------
```

```
      begin

         Answer(X) := Left(X) + Right(X);
         Answer(Y) := Left(Y) + Right(Y);
         Answer(Z) := Left(Z) + Right(Z);

         return Answer;

      end "+";

pragma PAGE;
      function "-" (Left  : Vectors;
                    Right : Vectors) return Vectors is

--       ----------------------
--       --declaration section-
--       ----------------------

         Answer : Vectors;

--    ----------------
--    --function body-
--    ----------------

      begin

         Answer(X) := Left(X) - Right(X);
         Answer(Y) := Left(Y) - Right(Y);
         Answer(Z) := Left(Z) - Right(Z);

         return Answer;

      end "-";

pragma PAGE;
      function Vector_Length (Vector : Vectors) return Elements is

      begin

         return Rsos(Vector(X), Vector(Y), Vector(Z));

      end Vector_Length;

pragma PAGE;
      function Dot_Product (Vector1 : Vectors;
                            Vector2 : Vectors) return Elements_Squared is

--       ----------------------
--       --declaration section-
--       ----------------------

         Answer : Elements_Squared;

--    ----------------
--    --function body-
--    ----------------
```

```
      begin

         Answer := Vector1(X) * Vector2(X) +
                   Vector1(Y) * Vector2(Y) +
                   Vector1(Z) * Vector2(Z);

        return Answer;

      end Dot_Product;

pragma PAGE;
      function Sparse_Right_Z_Add (Left  : Vectors;
                                   Right : Vectors) return Vectors is

--       ----------------------
--       --declaration section-
--       ----------------------

         Answer : Vectors;

--    ----------------
--    --function body-
--    ----------------

      begin

         Answer(X) := Left(X) + Right(X);
         Answer(Y) := Left(Y) + Right(Y);
         Answer(Z) := Left(Z);

         return Answer;

      end Sparse_Right_Z_Add;

pragma PAGE;
      function Sparse_Right_X_Add (Left  : Vectors;
                                   Right : Vectors) return Vectors is

--       ----------------------
--       --declaration section-
--       ----------------------

         Answer : Vectors;

--    ----------------
--    --function body-
--    ----------------

      begin

         Answer(X) := Left(X);
         Answer(Y) := Left(Y) + Right(Y);
         Answer(Z) := Left(Z) + Right(Z);

         return Answer;

      end Sparse_Right_X_Add;
```

```
pragma PAGE;
      function Sparse_Right_Xy_Subtract (Left  : Vectors;
                                         Right : Vectors) return Vectors is

--        ----------------------
--        --declaration section-
--        ----------------------

          Answer : Vectors;

--      ----------------
--      --function body-
--      ----------------

      begin

          Answer(X) := Left(X);
          Answer(Y) := Left(Y);
          Answer(Z) := Left(Z) - Right(Z);

          return Answer;

      end Sparse_Right_Xy_Subtract;

pragma PAGE;
      function Set_To_Zero_Vector return Vectors is

      begin

          return Zero_Vector;

      end Set_To_Zero_Vector;

   end Vector_Operations;

pragma PAGE;
   package body Vector_Scalar_Operations is

--      ----------------------
--      --declaration section-
--      ----------------------

      X : constant Axes := Axes'FIRST;
      Y : constant Axes := Axes'SUCC(X);
      Z : constant Axes := Axes'LAST;

pragma PAGE;
      function "*" (Vector     : Vectors1;
                    Multiplier : Scalars) return Vectors2 is

--        ----------------------
--        --declaration section-
--        ----------------------

          Answer : Vectors2;
```

```
--     --------------------
--     --begin function "*"
--     --------------------

       begin

          Answer(X) := Vector(X) * Multiplier;
          Answer(Y) := Vector(Y) * Multiplier;
          Answer(Z) := Vector(Z) * Multiplier;

          return Answer;

       end "*";

pragma PAGE;
       function "/" (Vector  : Vectors2;
                     Divisor : Scalars) return Vectors1 is

--        ----------------------
--        --declaration section-
--        ----------------------

          Answer : Vectors1;

--     --------------------
--     --begin function "/"
--     --------------------

       begin

          Answer(X) := Vector(X) / Divisor;
          Answer(Y) := Vector(Y) / Divisor;
          Answer(Z) := Vector(Z) / Divisor;

          return Answer;

       end "/";

pragma PAGE;
       function Sparse_X_Vector_Scalar_Multiply
                   (Vector     : Vectors1;
                    Multiplier : Scalars) return Vectors2 is

--        ----------------------
--        --declaration section-
--        ----------------------

          Answer : Vectors2;

--     ------------------------------------------------
--     --begin function Sparse_X_Vector_Scalar_Multiply
--     ------------------------------------------------

       begin

          Answer(X) := 0.0;
          Answer(Y) := Vector(Y) * Multiplier;
```

```
            Answer(Z) := Vector(Z) * Multiplier;
            return Answer;
        end Sparse_X_Vector_Scalar_Multiply;
    end Vector_Scalar_Operations;
end Coordinate_Vector_Matrix_Algebra;
```

```
separate (Coordinate_Vector_Matrix_Algebra)
package body Matrix_Operations is

-- ----------------------
-- --declaration section-
-- ----------------------

   X                : constant Axes     := Axes'FIRST;
   Y                : constant Axes     := Axes'SUCC(X);
   Z                : constant Axes     := Axes'LAST;

-- --the diagonal elements of Identity_Matrix will be set to 1.0 during
-- --package initialization
   Identity_Matrix : Matrices := (others => (others => 0.0));

   Zero_Matrix      : constant Matrices := (others => (others => 0.0));

-- --------------------
-- --subroutine bodies-
-- --------------------

   function Set_To_Identity_Matrix return Matrices is separate;

   function Set_To_Zero_Matrix return Matrices is separate;

pragma PAGE;
   function "+" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Matrices;

-- ----------------------------------------
-- --begin function "+" (matrices + matrices)
-- ----------------------------------------

   begin

      Answer(X,X) := Left(X,X) + Right(X,X);
      Answer(X,Y) := Left(X,Y) + Right(X,Y);
      Answer(X,Z) := Left(X,Z) + Right(X,Z);

      Answer(Y,X) := Left(Y,X) + Right(Y,X);
      Answer(Y,Y) := Left(Y,Y) + Right(Y,Y);
      Answer(Y,Z) := Left(Y,Z) + Right(Y,Z);

      Answer(Z,X) := Left(Z,X) + Right(Z,X);
      Answer(Z,Y) := Left(Z,Y) + Right(Z,Y);
      Answer(Z,Z) := Left(Z,Z) + Right(Z,Z);

      return Answer;

   end "+";
```

```
pragma PAGE;
   function "-" (Left  : Matrices;
                 Right : Matrices) return Matrices is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Matrices;

-- ----------------------------------------
-- --begin function "-" (matrices - matrices)
-- ----------------------------------------

   begin

      Answer(X,X) := Left(X,X) - Right(X,X);
      Answer(X,Y) := Left(X,Y) - Right(X,Y);
      Answer(X,Z) := Left(X,Z) - Right(X,Z);

      Answer(Y,X) := Left(Y,X) - Right(Y,X);
      Answer(Y,Y) := Left(Y,Y) - Right(Y,Y);
      Answer(Y,Z) := Left(Y,Z) - Right(Y,Z);

      Answer(Z,X) := Left(Z,X) - Right(Z,X);
      Answer(Z,Y) := Left(Z,Y) - Right(Z,Y);
      Answer(Z,Z) := Left(Z,Z) - Right(Z,Z);

      return Answer;

   end "-";

pragma PAGE;
   function "+" (Matrix : Matrices;
                 Addend : Elements) return Matrices is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Matrices;

-- ----------------------------------------
-- --begin function "+" (matrices + elements)-
-- ----------------------------------------

   begin

      Answer(X,X) := Matrix(X,X) + Addend;
      Answer(X,Y) := Matrix(X,Y) + Addend;
      Answer(X,Z) := Matrix(X,Z) + Addend;

      Answer(Y,X) := Matrix(Y,X) + Addend;
      Answer(Y,Y) := Matrix(Y,Y) + Addend;
      Answer(Y,Z) := Matrix(Y,Z) + Addend;

      Answer(Z,X) := Matrix(Z,X) + Addend;
```

```
      Answer(Z,Y) := Matrix(Z,Y) + Addend;
      Answer(Z,Z) := Matrix(Z,Z) + Addend;

      return Answer;

   end "+";

pragma PAGE;
   function "-" (Matrix     : Matrices;
                 Subtrahend : Elements) return Matrices is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Matrices;

-- ----------------
-- --function body-
-- ----------------

   begin

      Answer(X,X) := Matrix(X,X) - Subtrahend;
      Answer(X,Y) := Matrix(X,Y) - Subtrahend;
      Answer(X,Z) := Matrix(X,Z) - Subtrahend;

      Answer(Y,X) := Matrix(Y,X) - Subtrahend;
      Answer(Y,Y) := Matrix(Y,Y) - Subtrahend;
      Answer(Y,Z) := Matrix(Y,Z) - Subtrahend;

      Answer(Z,X) := Matrix(Z,X) - Subtrahend;
      Answer(Z,Y) := Matrix(Z,Y) - Subtrahend;
      Answer(Z,Z) := Matrix(Z,Z) - Subtrahend;

      return Answer;

   end "-";

pragma PAGE;
---------------------------------
--begin package Matrix_Operations-
---------------------------------

begin

-- --initialize diagonal elements of Identity_Matrix, remaining elements have
-- --already been set to 0.0

   Identity_Matrix(X,X) := 1.0;
   Identity_Matrix(Y,Y) := 1.0;
   Identity_Matrix(Z,Z) := 1.0;

end Matrix_Operations;
```

```
separate (Coordinate_Vector_Matrix_Algebra.Matrix_Operations)
function Set_To_Identity_Matrix return Matrices is

begin

    return Identity_Matrix;

end Set_To_Identity_Matrix;
```

```
separate (Coordinate_Vector_Matrix_Algebra.Matrix_Operations)
function Set_To_Zero_Matrix return Matrices is

begin

   return Zero_Matrix;

end Set_To_Zero_Matrix;
```

```
separate (Coordinate_Vector_Matrix_Algebra)
package body Matrix_Scalar_Operations is

-- ----------------------
-- --declaration section-
-- ----------------------

   X : constant Axes := Axes'FIRST;
   Y : constant Axes := Axes'SUCC(X);
   Z : constant Axes := Axes'LAST;

pragma PAGE;
   function "*" (Matrix     : Matrices1;
                 Multiplier : Scalars) return Matrices2 is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Matrices2;

-- ---------------------
-- --begin function "*"-
-- ---------------------

   begin

      Answer(X,X) := Matrix(X,X) * Multiplier;
      Answer(X,Y) := Matrix(X,Y) * Multiplier;
      Answer(X,Z) := Matrix(X,Z) * Multiplier;
      Answer(Y,X) := Matrix(Y,X) * Multiplier;
      Answer(Y,Y) := Matrix(Y,Y) * Multiplier;
      Answer(Y,Z) := Matrix(Y,Z) * Multiplier;
      Answer(Z,X) := Matrix(Z,X) * Multiplier;
      Answer(Z,Y) := Matrix(Z,Y) * Multiplier;
      Answer(Z,Z) := Matrix(Z,Z) * Multiplier;

      return Answer;

   end "*";

pragma PAGE;
   function "/" (Matrix  : Matrices2;
                 Divisor : Scalars) return Matrices1 is

--    ----------------------
--    --declaration section-
--    ----------------------

      Answer : Matrices1;

-- ----------------
-- --function body-
-- ----------------

   begin
```

```
      Answer(X,X) := Matrix(X,X) / Divisor;
      Answer(X,Y) := Matrix(X,Y) / Divisor;
      Answer(X,Z) := Matrix(X,Z) / Divisor;

      Answer(Y,X) := Matrix(Y,X) / Divisor;
      Answer(Y,Y) := Matrix(Y,Y) / Divisor;
      Answer(Y,Z) := Matrix(Y,Z) / Divisor;

      Answer(Z,X) := Matrix(Z,X) / Divisor;
      Answer(Z,Y) := Matrix(Z,Y) / Divisor;
      Answer(Z,Z) := Matrix(Z,Z) / Divisor;

      return Answer;

   end "/";

end Matrix_Scalar_Operations;
```

```
separate (Coordinate_Vector_Matrix_Algebra)
function Cross_Product (Left  : Left_Vectors;
                        Right : Right_Vectors) return Result_Vectors is

-- ----------------------
-- --declaration section-
-- ----------------------

   X      : constant Axes := Axes'FIRST;
   Y      : constant Axes := Axes'SUCC(X);
   Z      : constant Axes := Axes'LAST;

   Answer : Result_Vectors;

------------------------------
--begin function Cross_Product
------------------------------

begin

   Answer(X) := Left(Y) * Right(Z) - Left(Z) * Right(Y);
   Answer(Y) := Left(Z) * Right(X) - Left(X) * Right(Z);
   Answer(Z) := Left(X) * Right(Y) - Left(Y) * Right(X);

   return Answer;

end Cross_Product;
```

```
separate (Coordinate_Vector_Matrix_Algebra)
function Matrix_Vector_Multiply
          (Matrix : Matrices;
           Vector : Input_Vectors) return Output_Vectors is

-- ----------------------
-- --declaration section-
-- ----------------------

   X : constant Axes := Axes'FIRST;
   Y : constant Axes := Axes'SUCC(X);
   Z : constant Axes := Axes'LAST;

   Answer : Output_Vectors;

---------------------------------------
--begin function Matrix_Vector_Multiply
---------------------------------------

begin

   Answer(X) := Matrix(X,X) * Vector(X) +
                Matrix(X,Y) * Vector(Y) +
                Matrix(X,Z) * Vector(Z);

   Answer(Y) := Matrix(Y,X) * Vector(X) +
                Matrix(Y,Y) * Vector(Y) +
                Matrix(Y,Z) * Vector(Z);

   Answer(Z) := Matrix(Z,X) * Vector(X) +
                Matrix(Z,Y) * Vector(Y) +
                Matrix(Z,Z) * Vector(Z);

   return Answer;

end Matrix_Vector_Multiply;
```

```
separate (Coordinate_Vector_Matrix_Algebra)
function Matrix_Matrix_Multiply
          (Matrix1 : Left_Matrices;
           Matrix2 : Right_Matrices) return Result_Matrices is

-- ----------------------
-- --declaration section-
-- ----------------------

   X      : constant Axes := Axes'FIRST;
   Y      : constant Axes := Axes'SUCC(X);
   Z      : constant Axes := Axes'LAST;

   Answer : Result_Matrices;

-- ----------------
-- --function body-
-- ----------------

   begin

--     --first row

      Answer(X,X) := Matrix1(X,X) * Matrix2(X,X) +
                     Matrix1(X,Y) * Matrix2(Y,X) +
                     Matrix1(X,Z) * Matrix2(Z,X);

      Answer(X,Y) := Matrix1(X,X) * Matrix2(X,Y) +
                     Matrix1(X,Y) * Matrix2(Y,Y) +
                     Matrix1(X,Z) * Matrix2(Z,Y);

      Answer(X,Z) := Matrix1(X,X) * Matrix2(X,Z) +
                     Matrix1(X,Y) * Matrix2(Y,Z) +
                     Matrix1(X,Z) * Matrix2(Z,Z);

--     --second row

      Answer(Y,X) := Matrix1(Y,X) * Matrix2(X,X) +
                     Matrix1(Y,Y) * Matrix2(Y,X) +
                     Matrix1(Y,Z) * Matrix2(Z,X);

      Answer(Y,Y) := Matrix1(Y,X) * Matrix2(X,Y) +
                     Matrix1(Y,Y) * Matrix2(Y,Y) +
                     Matrix1(Y,Z) * Matrix2(Z,Y);

      Answer(Y,Z) := Matrix1(Y,X) * Matrix2(X,Z) +
                     Matrix1(Y,Y) * Matrix2(Y,Z) +
                     Matrix1(Y,Z) * Matrix2(Z,Z);

--     --third row

      Answer(Z,X) := Matrix1(Z,X) * Matrix2(X,X) +
                     Matrix1(Z,Y) * Matrix2(Y,X) +
                     Matrix1(Z,Z) * Matrix2(Z,X);

      Answer(Z,Y) := Matrix1(Z,X) * Matrix2(X,Y) +
                     Matrix1(Z,Y) * Matrix2(Y,Y) +
```

```
                    Matrix1(Z,Z) * Matrix2(Z,Y);

      Answer(Z,Z) := Matrix1(Z,X) * Matrix2(X,Z) +
                     Matrix1(Z,Y) * Matrix2(Y,Z) +
                     Matrix1(Z,Z) * Matrix2(Z,Z);

   return Answer;

end Matrix_Matrix_Multiply;
```

(This page left intentionally blank.)

SUPPLEMENTARY

INFORMATION

DEPARTMENT OF THE AIR FORCE
WRIGHT LABORATORY (AFSC)
EGLIN AIR FORCE BASE, FLORIDA, 32542-5434

ERRATA

AD-B120 255

REPLY TO ATTN OF: MNOI

13 Feb 92

SUBJECT: Removal of Distribution Statement and Export-Control Warning Notices

TO: Defense Technical Information Center
ATTN: DTIC/HAR (Mr William Bush)
Bldg 5, Cameron Station
Alexandria, VA 22304-6145

1. The following technical reports have been approved for public release by the local Public Affairs Office (copy attached).

	Technical Report Number	AD Number
1.	88-18-Vol-4	ADB 120 251
2.	88-18-Vol-5	ADB 120 252
3.	88-18-Vol-6	ADB 120 253
4.	88-25-Vol-1	ADB 120 309
5.	88-25-Vol-2	ADB 120 310
6.	88-62-Vol-1	ADB 129 568
7.	88-62-Vol-2	ADB 129 569
8.	88-62-Vol-3	ADB 129-570
9.	85-93-Vol-1	ADB 102-654
10.	85-93-Vol-2	ADB 102-655
11.	85-93-Vol-3	ADB 102-656
12.	88-18-Vol-1	ADB 120 248
13.	88-18-Vol-2	ADB 120 249
14.	88-18-Vol-7	ADB 120 254
15.	88-18-Vol-8	ADB 120 255
16.	88-18-Vol-9	ADB 120 256
17.	88-18-Vol-10	ADB 120 257
18.	88-18-Vol-11	ADB 120 258
19.	88-18-Vol-12	ADB 120 259

2. If you have any questions regarding this request call me at DSN 872-4620.

LYNN S. WARGO
Chief, Scientific and Technical
Information Branch

1 Atch
AFDTC/PA Ltr, dtd 30 Jan 92

ERRATA

DEPARTMENT OF THE AIR FORCE
HEADQUARTERS AIR FORCE DEVELOPMENT TEST CENTER (AFSC)
EGLIN AIR FORCE BASE, FLORIDA 32542-5000

REPLY TO
ATTN OF: PA (Jim Swinson, 882-3931) 30 January 1992

SUBJECT: Clearance for Public Release

TO: WL/MNA

The following technical reports have been reviewed and are approved for public release: AFATL-TR-88-18 (Volumes 1 & 2), AFATL-TR-88-18 (Volumes 4 thru 12), AFATL-TR-88-25 (Volumes 1 & 2), AFATL-TR-88-62 (Volumes 1 thru 3) and AFATL-TR-85-93 (Volumes 1 thru 3).

Virginia N. Pribyla

VIRGINIA N. PRIBYLA, Lt Col, USAF
Chief of Public Affairs

AFDTC/PA 92-039