Development of Radiations tolerant Attitude Control Algorithm for small Satellites

Supervisor

Prof. Reyneri LEONARDO

Candidate

Zubair Ahmed JUNAID

December 2020
Abstract

Small satellites provide flexible and economical cost-effective solutions for research and development in the field of the space industry. It is possible to implement and test various mission algorithms with very few resources available. The stringent requirement of a smaller mass of spacecraft has almost eliminated the use of propulsion systems in small satellites. The smaller mass and size of the spacecraft, on the other hand, make it easier to control and adjust the attitude and orbit of the spacecraft with only magnetic control or inertial control. In this field, there are many attitude and orbit control algorithms and techniques that have already been explored.

As compared to the heavier satellites, the smaller satellites are equally exposed to the radiations and experience the same harsh environment in space but due to the low mass budgeting, it is not preferred to use redundant hardware or extra shielding for them. Nowadays there are many circuits available that are made hardened against radiations during manufacturing. Integrated circuits, processors, and micro-controllers during the design phase can also implement hardware-level radiation hardening techniques to tolerate radiation exposure but this solution leads to very expensive and relatively slow hardware, which does not align with the main motivation of small satellites, i.e. small, inexpensive, and efficient.

In order to make the satellite to work while exposed to a radiation environment, software methods can be used. Once it is studied that how often and in which manner the radiations can affect the normal operations of spacecraft, a flexible and robust software algorithm can be developed which can tolerate not all, but many radiation events. This can lead to using cheap commercial off-the-shelf devices without using redundant hardware and heavy shielding. This technique may cause the system to use more memory and perform computation slowly, but it will be much faster and effective as compared to the hardware hardening techniques.

The work presented in this report comprised of developing a closed-loop Attitude control flight software for small satellites, such as CubeSat, along with spacecraft dynamics, and environmental effects. The flight software that interacts with the sensors and actuators mathematical models, can make the satellite work into any of the predefined mission modes. The flight software is made robust and hardened against space radiations which can alter the
normal software operations and can result in malfunctioning. The radiation effects are modeled in the software and are simulated. A comparative analysis is also performed to express how a radiation-tolerant flight software performs in presence of radiation effects as compared to non-hardened flight software.
Table of Contents

List of Tables ................................................................. IV
List of Figures ............................................................... V
Acronyms ........................................................................ VII
1 Introduction ................................................................. 1
  1.1 Aim of Study .......................................................... 1
  1.2 Proposed Solutions .................................................. 2
  1.3 Previous Work ........................................................ 2
  1.4 Thesis Organization ................................................... 3
2 Definitions and Notations ............................................. 4
  2.1 Reference Systems ................................................... 4
    2.1.1 Earth Centered Inertial (ECI) Frame ....................... 4
    2.1.2 Earth Centered Earth Fixed (ECEF) Frame ............... 4
    2.1.3 North East Down (NED) Frame ............................. 5
    2.1.4 Orbit Reference Frame ....................................... 6
    2.1.5 Body Frame ..................................................... 6
  2.2 Rotations ................................................................. 6
    2.2.1 Rotation Matrix ............................................... 7
    2.2.2 Euler Angles .................................................... 7
    2.2.3 Quaternions ..................................................... 8
  2.3 Space Radiations ..................................................... 9
    2.3.1 Types of Space Radiations ................................... 10
    2.3.2 Effects of Space Radiations ................................. 10
3 Attitude Control System ............................................ 12
  3.1 Mathematical Modeling .......................................... 12
List of Tables

3.1 Orbital Parameters Information .......................... 14
3.2 Gravitational Model Simulink Block Inputs & Outputs ... 16
3.3 Sun Position Model Simulink Block Inputs & Outputs .... 17
3.4 Magnetic Field Model Simulink Block Inputs & Outputs .. 18
3.5 Vehicle Dynamics Simulink Block Inputs and Outputs ... 21
3.6 Magnetometer Simulink Block Inputs and Outputs ...... 22
3.7 Gyroscope Simulink Block Inputs & Outputs ............ 23
3.8 Sun Sensor Simulink Block Inputs & Outputs ............ 25
3.9 GPS Simulink Block Inputs & Outputs ................... 25
3.10 Magnetorquer Rods Simulink Block Inputs & Outputs ... 27
3.11 Reaction wheels Simulink Block Inputs & Outputs ....... 30
3.12 Detumbling Mode Simulink Block Inputs & Outputs ..... 33
3.13 Sun Pointing Mode Simulink Block Inputs & Outputs ... 34
3.14 Nadir Pointing Mode Simulink Block Inputs & Outputs ... 36

4.1 List Of Data Structures in C++ Flight Software .......... 39

5.1 Detumbling mode performance comparison ................ 56
5.2 Nadir Pointing mode performance comparison ............. 57
5.3 Sun Pointing mode performance comparison ............... 58
List of Figures

2.1 ECI Coordinate System[3] .............................. 5
2.2 ECEF Coordinate System[3] .............................. 5
2.3 NED Coordinate System[4] .............................. 6
2.4 Orbit Reference frame and Body Coordinate System[2] 7
2.5 Quaternion Representation of rotation [6] 9

3.1 Attitude Control System Block Diagram .......... 13
3.2 Gravitational Forces acting on the Satellite .......... 16
3.3 Sun Vector Model ........................................ 17
3.4 Magnetic Field Model .................................... 18
3.5 Satellite Dynamics Simulink Library Block .......... 19
3.6 Magnetometer Simulink Block ......................... 23
3.7 Gyroscope representation of Simulink Block .......... 24
3.8 Sun Sensor representation of Simulink Block .......... 24
3.9 GPS representation of Simulink Block ................. 26
3.10 Magnetorquer Rods model Simulink representation 28
3.12 Reaction wheels representation of Simulink Model 29
3.13 Flight Software Top Level hierarchy in Simulink 31
3.14 Detumbling Mode Matlab/Simulink Block ............ 33
3.15 Sun Pointing Mode Simulink Model ................... 34
3.16 Nadir Pointing Mode Simulink Model ................. 35
3.17 Three-Dimensional Visualization of satellite in Orbit 37

4.1 Radiation Hardened Software Hierarchy ............... 42
4.2 S-function Builder Representation of Flight Software 44

5.1 Detumbling Mode Simulation Results: Angular rates along
xyz body axes .............................................. 46
5.2 Nadir Pointing Mode Attitude Error: Euler angles . . . . . . 47
5.3 Nadir Pointing Mode Attitude Error steady state: Euler angles 48
5.4 Sun Pointing Mode Attitude Error: Euler angles . . . . . . . 49
5.5 Sun Pointing Mode Attitude Error steady state: Euler angles 49
5.6 Radiations affected Detumbling Mode Simulation Results:
  Angular rates along xyz body axes . . . . . . . . . . . . . . . 50
5.7 Radiations affected Nadir Pointing Mode Simulation Results:
  Euler angles . . . . . . . . . . . . . . . . . . . . . . . . . . . . 51
5.8 Radiations affected Sun Pointing Mode Simulation Results:
  Euler angles . . . . . . . . . . . . . . . . . . . . . . . . . . . . 52
5.9 Radiation Hardened Detumbling Mode Simulation Results:
  Angular rates along xyz body axes . . . . . . . . . . . . . . . 53
5.10 Radiation Hardened Nadir Pointing Mode Simulation Results:
  Euler angles . . . . . . . . . . . . . . . . . . . . . . . . . . . . 54
5.11 Radiation Hardened Nadir Pointing Mode steady state Simula-
  tion Results: Euler angles . . . . . . . . . . . . . . . . . . . . 55
5.12 Radiation Hardened Sun Pointing Mode Simulation Results:
  Euler angles . . . . . . . . . . . . . . . . . . . . . . . . . . . . 56
5.13 Radiation Hardened Sun Pointing Mode steady state Simula-
  tion Results: Euler angles . . . . . . . . . . . . . . . . . . . . 57
Acronyms

ECI
Earth Centered Inertial co-ordinate Frame

ECEF
Earth Centered Earth fixed co-ordinate Frame

NED
North East Down co-ordinate Frame

ORF
Orbit Reference Frame

PID
Proportional-Integral-Derivative

LEO
Low Earth Orbit

SEU
Single Event Upset
Chapter 1

Introduction

For more than the past decade, small satellites known as CubeSats and nanosats have become an area of research and development. The enhancements in the scientific equipment of satellite payload, as well as the platform, have made the possibility of size and mass reduction of the satellite. Many academic institutes and commercial companies are doing research and experimenting with new innovations in terms of software algorithms and also in terms of hardware. In terms of orbit and attitude control of the satellite, due to the reduced mass and volume, the forces and torques needed to control the position and three-dimensional orientation of the satellite have also reduced. This has led to the almost elimination of propulsion systems from small satellites, leaving the control only to the magnetic or inertial control. Many techniques have been developed and explored in this regard. Unfortunately in space, despite having different sizes and masses, the space radiations act on all satellites the same way. It can cause transient bit flipping to permanent fault induction in hardware and memory element or can cause short-circuit or open-circuit resulting in malfunctioning of a device or set of devices or in the worst case the mission failure. Also, the longer exposure to radiation can tear-off the parts of satellites causing physical damage to the spacecraft.

1.1 Aim of Study

The work presented in this report is focused on the development of a basic attitude and orbit control algorithm which can withstand radiation effects. The study aimed at the development of a closed-loop mathematical model in order to simulate a set of control algorithms for a complete mission of a
small satellite. The other part of the study focused on making these control algorithms radiation tolerant. Then the comparative analysis is performed between hardened flight software and un-hardened flight software against radiation effects.

1.2 Proposed Solutions

With the help of the MathWorks modeling tool, Matlab/Simulink, it is possible to model and simulate the attitude and orbit subsystem of a satellite. The attitude control subsystem with different environmental effects can be simulated along with sensors and actuators’ mathematical models. The satellite can be made to point to different targets or to damp its angular rotations based on the mission control modes. The whole system can be visualized in three-dimensional space with the help of a virtual reality add-on available in Matlab/Simulink. The flight software model can be converted into C or C++ language in order to model and analyze the effects of radiations. Some hardening techniques can be used to make the flight software hardened and more robust against radiation effects. The hardened software can be plugged back into the Simulink model by using the s-function capability of Matlab which allows a low-level language developed function or set of functions to run as a part of the Simulink model. The comparative analysis can be performed in the end to evaluate the study and simulation results.

1.3 Previous Work

There is an already developed C++ library that implements the triple redundancy on the level of the variables. It stores three private copies of each defined data type variable and over-rides the basic operators, providing a seamless view to the user. The ideology is such that during the radiation event, it is common that Single Event Upsets happen. This phenomenon not only cause the bit-flipping in memories but also can affect the execution of program by corrupting the control variables or any vital data. Defining three copies of same variable will most probably cause the corruption of only one version of stored copy, leaving the other two versions unchanged. Upon reading the variable, a voting is performed and the correct version of data is retrieved [1].
Defining the triple data can be seen as:

\[
\text{HardenedData} :: \text{TripleData} < \text{DataType} > \text{Variable};
\]  

(1.1)

Accessing any of the private copy is not the goal of normal software operations, but can be done for the sake of radiation noise injection in the following way:

Variable.a();
Variable.b();
Variable.c();

There is also another Library provided, written in C++ language, with the name of SingleData.cpp, that overrides the basic operations just like triple data, but keeps only one copy of the data. The purpose of this file is to provide a flexibility at the development level that the user can choose to use radiation hardened software or un-hardened software in order to analyze the effects of radiations.

1.4 Thesis Organization

Chapter 1 discusses the brief introduction to the topic with the aim defined to conduct this study. A short summary of the proposed solution is also mentioned which is discussed in further detail in the following chapters. The previous work done which is used as a starting point of this thesis is also described.

Chapter 2 discusses the basic notations and definitions and concepts used in the whole study. A basic definition of various coordinate transformations and rotation matrices is discussed. The concept of space radiations is also discussed.

Chapter 3 discusses the Attitude Control System in detail with the description of Matlab/Simulink various blocks. The flight software is also presented in detail.

Chapter 4 discusses the methodology for the radiation hardening of control algorithms. Development platform and building blocks of Radiations hardening software are also presented.

Chapter 5 discusses the simulation results and concludes the comparative analysis of radiation-tolerant and intolerant software. This chapter concludes the study and highlights all the results achieved and summarizes them.

Chapter 6 discusses the conclusion of the report and the presented work.
Chapter 2
Definitions and Notations

2.1 Reference Systems

Reference systems are extremely important when defining and studying the space craft motion and orientation in space. There are numerous co-ordinate frames used in the report which are summarized below.

2.1.1 Earth Centered Inertial (ECI) Frame

This is an inertial reference frame with the origin lies at the center of earth. The x-y plane is the equatorial plane with the z-axis points towards the north pole of earth.

The x-axis points towards the vernal equinox and y-axis completes the right-handed Cartesian coordinate system. The nutation and precession motions of earth are not considered in this report for the sake of simplicity.

2.1.2 Earth Centered Earth Fixed (ECEF) Frame

Earth centered Earth fixed frame, as the name suggests, rotates with the rotation of earth. The origin is again at the center of the earth, the z-axis points towards the North pole and is aligned with the rotational axis of earth.

The x-axis points toward the intersection of Greenwich meridian with the equator. The y-axis completes the right handed system.
2.1.3 North East Down (NED) Frame

The North East Down coordinate system defines a reference frame on the body of spacecraft. The x-axis points parallel towards the north pole of earth, the y-axis points parallel towards the latitude curve of earth and z-axis points towards the center of earth, completing the right hand rule.
2.1.4 Orbit Reference Frame

The origin of Orbit reference frame lies on the center of mass of space craft, while z-axis points towards the center of the Earth. X-axis points in the direction of motion tangentially to the orbit [2].

2.1.5 Body Frame

The Body coordinate frame moves and rotates with the body movement. The origin lies on the center of mass of satellite. The z-axis points towards the face which is usually required to face towards earth during earth-pointing mode, where the scientific equipment of a LEO satellite is placed. The x-axis points along the solar panels, considering the deploy-able solar panels. The y-axis completes the right-hand rule.

2.2 Rotations

In order to represent the orientation of space craft in different frame of references and to perform different operations on vectors that are described in different frames of reference, the rotations of vectors is necessary.
2.2.1 Rotation Matrix

A rotation matrix is a matrix that represents the rotations of two frame of references in Euclidean Space. The rotation of a vector from one frame to another frame can be performed as follows:

\[ v^y = R_x \times v^x \]  \hspace{1cm} (2.1)

2.2.2 Euler Angles

The successive angular rotations about the three orthogonal body axes are defined as the Euler angle rotations\^[2]. Roll(\(\phi\)), pitch(\(\theta\)) and yaw(\(\psi\)) are respectively the rotations around x, y and z axis.

The rotation around x-axis can be seen as the x-axis component of source and target vector does not change,

\[ R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos \phi & \sin \phi \\ 0 & -\sin \phi & \cos \phi \end{bmatrix} \]  \hspace{1cm} (2.2)

The rotation around y-axis, which is called pitch, can be seen as the y-axis component of source and transformed vector does not get affected.

\[ R_y(\theta) = \begin{bmatrix} \cos \theta & 0 & -\sin \theta \\ 0 & 1 & 0 \\ \sin \theta & 0 & \cos \theta \end{bmatrix} \]  \hspace{1cm} (2.3)
When the rotation around z-axis is performed, which is called yaw, can be seen in the following equation as the z-axis component of source and target vector remains unchanged.

$$R_z(\psi) = \begin{bmatrix} \cos \psi & -\sin \psi & 0 \\ \sin \psi & \cos \psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

(2.4)

### 2.2.3 Quaternions

Quaternions are an alternate way to describe orientation and rotations in three dimensional space using a set of four numbers. They have the ability to uniquely describe any three-dimensional rotation about an arbitrary axis and do not suffer from gimbal lock [5].

Quaternions are represented as:

$$q = q_1i + q_2j + q_3k + q_4$$

(2.5)

It is convenient and preferable to define the attitude of the satellite using quaternions. It is a generalized complex number, composed of the sum of a real number $q_4$ and a 3D vector $q_{1-3}$ [3].

The closest approximation of understanding that can be developed to grasp the concept of quaternions is that it is a rotation between two reference frames on an arbitrary axis. The scalar part represents the amount of rotation which is the angle, and the vector part represents the axis of rotation.

An important property of the quaternions is that sum of squares of all four components is always 1 [3].

$$q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1$$

(2.6)

The quaternions can be used to transform one quaternion orientation to another quaternion using multiplying them together. For example, quaternion rotation between reference frame 3 and 1 can be represented as multiplying the quaternion rotations that represent rotations between 3 to 4 and 4 to 1.

$$q_{31} = q_{34}q_{41}$$

(2.7)

During the pointing control modes of space craft, the attitude error between the reference attitude quaternion and the current attitude quaternion is also
calculated in the similar way.

$$q_{error} = q_{ref} q_{attitude}$$  \hspace{1cm} (2.8)

### 2.3 Space Radiations

Space weather is a phenomenon of effects of solar activity on the near-earth space environment. The energized charged particles can not only bombard the exposed surfaces of satellites but also can penetrate through sometimes hardware devices and circuits and cause hardware and software malfunctioning. The unexplainable errors in functions of sensitive parts and also the degradation of structural materials are known effects of harsh space weather[7].
2.3.1 Types of Space Radiations

There are three types of space radiations widely considered for space craft modeling.

The Trapped Particles

The charged particles from solar wind are trapped in the earth’s surrounding Van Allen radiation belts and travel along the lines of magnetic field. The inner belt contains protons with energy exceeding 10 MeV, while the outer belt mostly contains charged electrons. The satellites experience the radiation effects caused by these trapped particles[7].

The Transient Particles

The transient particles are mainly the reason of radiation events such as solar wind, solar flares, coronal mass ejections (CME)[7]. The large scale disturbances caused by transient particles can impact on space crafts normal operations.

Galactic Cosmic Radiation

These radiations originate from outside solar system. They consist of ionized atoms ranging from a single proton up to a uranium nucleus[7]. The flux of this radiation type is low, Earth’s magnetic field provides natural shielding against it. Therefore, as the altitude increases, the exposure of GCRs increases.

2.3.2 Effects of Space Radiations

When a satellite is injected into its orbit, the space weather affects its normal mission life in various ways. The most common effects are:

- **spacecraft charging**: It can be surface charging which can cause structural damage to the space craft or it can be internal dielectric charging which might cause the abnormal operations and damage of sensitive electronics equipment.

- **Total Ionizing Dose**: It defines the total energy that ionization processes create and deposit in materials when energized particles pass
through it [7]. It can degrade the structure of satellite and can cause mission failure also.

• **Displacement Damage**: An incoming energetic particle may collides with the lattice atoms and can result in atoms displacement. The properties of bulk semiconductor material may alter which might cause the silicon level defect.

• **Single Event Effects**: It is the effect when a single incident ionizing particle depositing enough energy to cause an abnormal effect[7]. There are four types of single event effects:
  
  – *SEU*: Single Event Upset
  – *SEL*: Single Event Latch-up
  – *SEB*: Single Event Burn-out
  – *SEGR*: Single Event Gate Rupture

SEUs result in change of logic state of a part of sensitive micro-electronic device. It can result in bit flipping of some part of data while it is residing in the memory or during execution. This can cause the software to consider the wrong values of data and can result in program corruption. An even more severe case of SEU is Single Event Function Interrupt (SEFI), which can affect the control of software and can put the system in test mode, halt or any undefined state. This will then require a full reset of the system.

In this report, the SEU bit flipping is mainly targeted, modeled and treated.
Chapter 3
Attitude Control System

The three dimensional orientation of spacecraft in space is termed as attitude of the satellite. Having continuous, or in some cases periodic knowledge, and having the ability to control it under desired thresholds is of vital importance to every satellite mission. Few main purpose of attitude controlling of a spacecraft can be described as:

- To point the scientific equipment of satellite in right direction (under threshold desired accuracy) so that mission objectives can be successfully achieved.

- To keep the sensitive sensors and scientific equipment to face the direct sunlight and in some cases direct earth to avoid the earth albedo effect, so that they might not get damaged.

- To keep pointing the solar panels towards sun as maximum as possible in order not to drain out the batteries power.

3.1 Mathematical Modeling

The Attitude Control System is modeled and simulated through Mathworks Matlab/Simulink mathematical modeling tool. It provides a powerful platform to not only simulate the mathematical equations for spacecraft but also it can run the C/C++ software codes completely integrated with the Simulink blocks and seamless to the user. The extensive Simulink library provides various blocks with plug-n-place capability to make the modeling easier.
A closed-loop CubeSat simulation library is provided with the Matlab/Simulink Aerospace Blockset which lets the user to model, simulate, analyze, and visualize the motion and dynamics of CubeSats and nano satellites, which are miniaturized spacecraft designed for space research[8].

A simplified block diagram of the Attitude Control System presented in this report is given. The details of different blocks of Simulink used for closed loop simulation of Attitude Control System are described in this chapter.

### 3.1.1 Mission Description

In the work presented here, a LEO satellite mission with a general imaging payload is considered. For each of such kind of missions, three basic maneuvers or attitude controls are required to be implemented for the completion of basic mission functionalities. The following working modes are considered in this work.

- **Detumbling Mode:** When the satellite is ejected from the launcher and is injected into its orbit, the initial angular velocities in each of its body axes with respect to any of the inertial frame of references are very large and sometimes in the order of 10’s of degrees per second. It is very crucial to implement a control law that can help slow down the body angular velocities under some threshold. In this study, this control mode is implemented with the name of *Detumbling Mode*, that relies on the
magnetometers and magnetorquer rods to damp the satellite body rates. Another purpose of this mode is to *de-saturate* the reaction wheels of satellite because when the reaction wheels are required to slow down their speed, they induce the angular torques in the body which are also damped via this mode.

- **Sun Pointing Mode:** To keep charging the batteries of satellite, it is necessary to keep pointing the solar panels towards the sun. Therefore, a specific pointing mode is designed in which satellite points its face, that contains the solar panels, towards the sun. In the study presented here, the sun pointing mode is considered as a separate mission mode which relies on the sun sensors and attitude information, and gives commands to reaction wheels in order to point towards the sun.

- **Nadir Pointing Mode:** The mission mode in which satellite is required to point its scientific payload towards its target is a basic necessary mission mode. In the study presented here, the satellite is made to point towards the center of earth. This mode relies on the GPS data and attitude information, and controls the attitude using reaction wheels.

### 3.1.2 Orbital Parameters

The mission is considered as a general LEO satellite in order to simulate and understand the attitude control software and radiation hardening implementation. The summary of mission geometry can be sum up as following:

<table>
<thead>
<tr>
<th>Keplerian Orbital Elements</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semi-major Axis (a)</td>
<td>6786233.13</td>
</tr>
<tr>
<td>Eccentricity (e)</td>
<td>0.0010537</td>
</tr>
<tr>
<td>Inclination (i)</td>
<td>51.7519</td>
</tr>
<tr>
<td>Right Ascension of Ascending Node (Ω)</td>
<td>95.2562</td>
</tr>
<tr>
<td>Argument of periapsis (ω)</td>
<td>93.4872</td>
</tr>
<tr>
<td>True Anomaly (v)</td>
<td>302.9234</td>
</tr>
<tr>
<td>Epoch Date &amp; Time</td>
<td>4-January-2019-12:00</td>
</tr>
</tbody>
</table>

*Table 3.1: Orbital Parameters Information*
3.2 Environment

The first component of closed-loop simulator is the environment model which simulates the environment effects that act on the satellite at each time stamp. These blocks are not directly the part of Attitude Control system, but they are necessary for the system to work in closed loop manner. The environmental forces and torques acting on the body along with the magnetic field intensity and sun position model are the main components of environmental effects modeling.

3.2.1 Gravitational Model

The aerospace blockset provides a Spherical Harmonic Gravity Model that implements the mathematical representation of spherical harmonic gravity based on the earth gravitational potential. It provides a convenient way to describe the earth’s gravitational field outside of its surface in spherical harmonic expansion[9].

The gravitational force acting on the satellite can be described as:

\[ F = G \frac{M m}{r^2} \]  

where \( G \) is the gravitational constant and its value is \( 6.67430 \times 10^{-11} \frac{Nm^2}{kg^2} \). This block takes the position of satellite at each instant in ECEF coordinate frame and calculates the gravity values in ECEF coordinate frame. These values are then multiplied with the mass of satellite and transformed into the body frame in order to represent the gravitational force acting on the satellite in Body Coordinate frame, according to the Newton’s second law of motion. The precession of the earth is ignored for the sake of simplicity.

\[ F = ma \]  

The Simulink block of Gravitational Model’s inputs and outputs are given below.

3.2.2 Sun Position Model

The sun model calculates the sun vector at each time stamp provided. The model is part of the aerospace block-set celestial phenomenon. The input is
### 3.2.3 Magnetic Field Model

The magnetic field model is also the part of aerospace block-set environment. It implements the 12th degree International Geomagnetic Reference Field model. The inputs are the position of space craft in terms of latitude, longitude and altitude. Also the start date time is also required to be provided. The output, which is of interest, is the Magnetic Field intensity vector in NED coordinate frame. The output unit is nano-tesla.

The Simulink block of Magnetic Field Model’s inputs and outputs are given below.

### 3.2.4 Disturbances

When the satellite is injected into its orbit, it experiences several environmental disturbances. These disturbances in some cases must be considered and modeled properly in order to calculate the proper torque commands for actuators. In this study, for the sake of simplicity of model and unit inertia
matrix consideration, the disturbances are not taken into account. Generally, the environmental disturbances that can act on the body of space craft are following:

**Gravitational Torque**

Gravitational force acting on the satellite body causes a gravitational torque on the body when its mass is not symmetrically distributed. This non-zero torque is due to the different amount of gravitational force acting on the different parts of satellite. It can be modeled as[10]:

\[ T_G = (F_2 - F_1)\frac{l}{2} \sin \alpha \]  

(3.3)

**Aerodynamic torque**

The air density at higher altitude start to lower. Some LEO satellites may face some drag but it is in quite lower amount. The drag force acting on the satellite body can be modeled as[10]:

\[ F_{AD} = \frac{\rho C_d A V^2_o}{2} \]  

(3.4)
### Magnetic Field Model Simulink Block Inputs & Outputs

<table>
<thead>
<tr>
<th>Input</th>
<th>Unit</th>
<th>Output</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lat-Lon-Alt (Lat., Lon., Alt.)</td>
<td>deg, deg, meter</td>
<td>Bned (Magnetic Field Intensity Vector in NED Frame)</td>
<td>Tesla</td>
</tr>
</tbody>
</table>

Table 3.4: Magnetic Field Model Simulink Block Inputs & Outputs

![Magnetic Field Model Simulink Block](image)

**Figure 3.4: Magnetic Field Model**

### Magnetic Torque

The scenario when the spacecraft has non-zero magnetic moment \( m \) due to hysteresis and other currents flowing, the external magnetic field flux \( B \) causes a torque to be produced in the body.

\[
\vec{T}_m = \vec{m} \times \vec{B} 
\]

### Solar Radiation Pressure

The solar radiation pressure, as the name suggests, is the force of solar radiations acting on the body. This force, when acting on a moment arm of satellite, creates a torque on the body.

\[
\vec{T}_s = r_{sp} \times \vec{F}_s \vec{F}_s = (1 + K) \rho_s A_\perp 
\]

### 3.3 Satellite Dynamics

Simulink provides a powerful library block for the analysis of satellite dynamics equation and calculations of various transformation matrices. The aerospace blockset equations of motions’ 6DOF (6 Degrees of Freedom) represents the orientation of satellite in space using quaternions representation.
defined in ECEF frame. The block takes the initial position, velocity, orientation, and angular rates along with the spacecraft mass and inertia as the initial conditions. Then it propagates the output vectors at each time stamp based on the previous states. The block also takes the forces and torques acting on the satellite from the environment, and also the torques acting on the body from the actuators. This block provides the inputs for

Figure 3.5: Satellite Dynamics Simulink Library Block

the sensors and after few transformations, the quaternions representing the attitude of spacecraft can also be computed. The quaternion angles from body to ECEF frame and from body to ECI are also computed. Moreover, the rotation matrix from body to NED frame is also possible to calculate. The rates in the principal axes of body are also calculated.
The inputs and the outputs generated by this block which are used by further blocks are summarized below:
## Attitude Control System

<table>
<thead>
<tr>
<th>Input</th>
<th>Unit</th>
<th>Output</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forces</td>
<td>N (Newton)</td>
<td>Vecef</td>
<td>m/s (meters/second)</td>
</tr>
<tr>
<td>(Environmental Forces acting on the body)</td>
<td></td>
<td>(3×1 Vector. Velocity of Satellite in ECEF Frame)</td>
<td></td>
</tr>
<tr>
<td>Torques</td>
<td>N-m (Newton meter)</td>
<td>Xecef</td>
<td>m (meters)</td>
</tr>
<tr>
<td>(Environmental Torques and Actuators Torques acting on the body)</td>
<td></td>
<td>(3×1 Vector. Position of satellite in ECEF Frame)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>LatLonAlt</td>
<td>Deg,Deg &amp; meters</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(3×1 Vector. Latitude, Longitude &amp; Altitude of satellite)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DCM_{bi}</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(3×3 Vector. Direction Cosine Matrix between ECI to Body Frame)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DCM_{be}</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(3×3 Vector. Direction Cosine Matrix between ECI to NED Frame)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>DCM_{ef}</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(3×3 Vector. Direction Cosine Matrix between ECEF to NED Frame)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>(\omega_{rel})</td>
<td>rad/sec (radians per second)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(3×1 Vector. Angular rates of the body with respect to NED Frame)</td>
<td></td>
</tr>
</tbody>
</table>

**Table 3.5:** Vehicle Dynamics Simulink Block Inputs and Outputs
3.4 Sensors

Sensors are the eyes and ears of a space craft for controlling and estimating the attitude of satellite. There are few sensors that are modeled in this work that support the attitude control system. The purpose of sensor models is to represent a close to reality mathematical model which incorporates the noises and irregularities of sensor’ readings along with the necessary axes transformations.

Actual sensors’ model needs to take input as the physical stimuli quantities and gives the outputs which represent the real sensors’ outputs like currents or voltages. Then, it should be the job of flight software to take current or voltage signals from sensors’ model and perform ADC conversions to get the digital quantized readings of sensors and calculate the vectors and matrices in respective coordinate frames. But, due to the sake of simplicity and limited scope of this work, the sensors’ models provide the vectors and matrices in respective coordinate frames.

3.4.1 Magnetometer

The magnetometer takes the magnetic field being calculated from environment block and transform it into body axes. It also induces the noises and bias that is faced by the real sensor. The sensor model is supposed to give magnetic field values in Body coordinate frame.

The Simulink representation of Magnetometer is shown in figure.

<table>
<thead>
<tr>
<th>Input</th>
<th>Unit</th>
<th>Output</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>MagEarth</td>
<td>Tesla</td>
<td>$B_{MGM}$</td>
<td>Tesla</td>
</tr>
<tr>
<td>(3 × 1 Magnetic Field Intensity Vector in NED Frame)</td>
<td></td>
<td>(3 × 1 Magnetic Field Intensity Vector in Body Frame)</td>
<td></td>
</tr>
<tr>
<td>NED2Body</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(3 × 3 NED to Body transformation Vector)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.6: Magnetometer Simulink Block Inputs and Outputs
3.4.2 Gyroscope

The purpose of gyroscope is to measure the angular velocities, which are also called body rates, in the body coordinate frame of satellite. In order to simulate the near-to-real effects of gyroscope, the colored noise model for bias is added with the readings coming from the dynamics block ($\omega_{rel}$).

<table>
<thead>
<tr>
<th>Input</th>
<th>Unit</th>
<th>Output</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\omega$</td>
<td>rad/sec</td>
<td>$\omega_{SENS}$</td>
<td>rad/sec</td>
</tr>
<tr>
<td>(3 × 1 Angular velocity in body axes of satellite)</td>
<td>(radians per second)</td>
<td>(3 × 1 Angular velocity in body axes affected by noises)</td>
<td>(radians per second)</td>
</tr>
</tbody>
</table>

Table 3.7: Gyroscope Simulink Block Inputs & Outputs

3.4.3 Sun Sensor

The purpose of sun sensor is to give the sun position vector in body coordinate frame of reference. In order to develop a mathematical model for the sun
sensor, the sun position vector at the given time stamp is generated by sun position model and added with bias noises to represent the more realistic measurement model.

Figure 3.8: Sun Sensor representation of Simulink Block
### Attitude Control System

<table>
<thead>
<tr>
<th>Input</th>
<th>Unit</th>
<th>Output</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>xSunECI (3×1 Sun Position Vector in ECI Frame)</td>
<td>m (meters)</td>
<td>xSunECISensor (3×1 Sun Position Vector in ECI Frame affected by noises)</td>
<td>m (meters)</td>
</tr>
</tbody>
</table>

**Table 3.8:** Sun Sensor Simulink Block Inputs & Outputs

### 3.4.4 GPS

The main goal of Global Positioning System sensor is to generate position and velocity in ECEF frame of reference. The same is generated and taken as an input from the dynamics block. There is also an orbit propagator block, by simulink aerospace blockset, used in parallel in order to keep correcting the values of position and velocity generated by dynamics block or to make selection between *Orbit Propagator* or Dynamics Block position & velocity data. The orbit propagator block takes the epoch date of simulation and orbital parameters to propagate the position and velocity at every time stamp. Then, noise is added with the readings in order to represent the more realistic scenario.

The Simulink representation of *GPS* model is shown in figure. The *zero check* block makes sure that on the first instant of simulation, the zeros may not propagate to the next blocks.

<table>
<thead>
<tr>
<th>Input</th>
<th>Unit</th>
<th>Output</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>xECEF (3×1 Position Vector in ECEF Frame)</td>
<td>m (meters)</td>
<td>xECEFGPS (3×1 Position Vector in ECEF Frame affected by noises)</td>
<td>m (meters)</td>
</tr>
<tr>
<td>vECEF (3×1 Velocity Vector in ECEF Frame)</td>
<td>m/s (meters per second)</td>
<td>vECEFGPS (3×1 Velocity Vector in ECEF Frame affected by noises)</td>
<td>m/s (meters per second)</td>
</tr>
</tbody>
</table>

**Table 3.9:** GPS Simulink Block Inputs & Outputs

25
3.5 Actuators

Actuators help in representing a mathematical approach to apply the required torque on body axes, which is calculated by the flight software in order to control the attitude of space craft or in order to damp the angular rates of satellite. In small satellites, unlikely the bigger satellite, the propulsion systems are not used in order to keep the mass budget lower. Thus, the other actuation methods although provide the necessary actuation, but are slower and less effective as compared to the propulsion system.

In real, actuators mathematical models should take the values of physical electrical stimuli like currents or voltages and calculate the respective torque commands. The conversion from torque to electrical stimuli should be taken into account by the flight software, but for the sake of simplicity and limited scope of the work, the actuators models take the torque or dipole vectors and performs calculations.
3.5.1 Magnetorquer Rods

The magnetorquer rods consist of a coil wound on ferromagnetic core or an air core. When the rod is energized (current is passed through the coil), it generates a magnetic dipole moment which depends on the number of turns of coil, the current passing through the coil and area of coil.

\[ \vec{M} = NI\vec{A} \]  

(3.7)

If that magnetic dipole is not aligned with earth’s magnetic field, then a torque is produced. The resultant torque can be computed simply by taking the cross product of two elements.

\[ \vec{\tau} = \vec{M} \times \vec{B} \]  

(3.8)

In the study presented in this report, in order to have a control in all three axes of space craft, three magnetorquer rods are considered to be placed orthogonal to each other. The flight software generates a set of Dipole Commands required to be generated by the torquer rods which are mapped on each of the magnetorquer rod. The magnetic field vector in Body frame \( B_{\text{Body}} \) is multiplied with the alignment matrix of each magnetorquer rod to compute the torque generated by each rod in body axes.

<table>
<thead>
<tr>
<th>Input</th>
<th>Unit</th>
<th>Output</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>DipoleCMD (3 × 1 Dipole Commands for each of three magnetorquer rods)</td>
<td>( A - m^2 ) (Ampere-square-meter)</td>
<td>Torque (3 × 1 Torque vector applied on each of body axis)</td>
<td>N-m (Newton-meter)</td>
</tr>
<tr>
<td>( B_{\text{BODY}} ) (3×1 Magnetic Field Intensity vector in Body frame)</td>
<td>Tesla</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 3.10:** Magnetorquer Rods Simulink Block Inputs & Outputs
3.5.2 Reaction Wheels

Reaction wheels are the spinning wheels mounted on a motor placed on the satellite. Their speed can be increased and decreased as per requirement of resultant torque. A reaction wheel can provide torque on its axis of rotation. A tetrahedron configuration is assumed in this work that can provide the torque in any of three axes of satellite body. There are four reaction wheels assumed to be placed in tetrahedron configuration and they can withstand with one wheel failure. The advantage of this configuration is that the wheel assembly can provide twice as much of maximum torque on an axis that a single wheel can provide\cite{2}. The tetrahedron configuration is shown in figure. The homogeneous distribution matrix of Reaction wheels in tetrahedron configuration is shown.

\[
L = \begin{bmatrix}
\sqrt{3}/3 & -\sqrt{3}/3 & -\sqrt{3}/3 & \sqrt{3}/3 \\
\sqrt{3}/3 & -\sqrt{3}/3 & \sqrt{3}/3 & -\sqrt{3}/3 \\
\sqrt{3}/3 & \sqrt{3}/3 & -\sqrt{3}/3 & -\sqrt{3}/3
\end{bmatrix}
\] (3.9)
The flight software generates a set of torque commands in body frame to be applied on satellite for the reaction wheels. These torque commands are mapped on the tetrahedron configuration of wheels by multiplying with distribution matrix. It is also assumed in this study, that the maximum torque that each wheel can produce is $5\text{mN} - m$, which is a common value for the commercially available reaction wheels for Cubesats. Therefore, the torque is limited to this value using saturation block and finally mapped back on to the body frame.

Figure 3.11: Tetrahedron configuration of Reaction Wheels[2]

Figure 3.12: Reaction wheels representation of Simulink Model
### De-Saturation of Reaction Wheels

The principle of reaction wheel is to transfer the total angular momentum of satellite between body and reaction wheels. The acceleration or deceleration can not be altered in an unlimited amount. The wheels, once accelerated and reached to their maximum speed, can not produce any more torque and are considered saturated. Therefore, a de-saturation of angular momentum of reaction wheels is needed. This action is also performed using magnetorquer rods. The torquer rods are required to produce torque in such axes so that they can off-load the reaction wheels and their speed can be decreased. This function is not modeled in this work and not presented in the mathematical modeling.

### 3.6 Flight Software

The flight software is the main Attitude Control software. It is the algorithm that takes the information of attitude and different required vectors from sensors and dynamics blocks and calculates the required torque to be applied on the actuators in order to maintain or control the attitude of satellite. This is the part of mathematical model which is then transformed into piece of software and is embedded in to the micro-controller or processor. The simplicity and execution efficiency of this block is of utmost concern. The output of this block is the required torque in body frame to control the orientation of space craft. The required commands are fed to the actuators and then the actuators block decides that how the torque will be acting on the body. There are three different mission modes considered in this work in order to better describe the functionality of attitude control software.

- Detumbling Mode

<table>
<thead>
<tr>
<th>Input</th>
<th>Unit</th>
<th>Output</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>TorqueCMD (3 x 1 Torque Commands required to be applied on body axes)</td>
<td>$N \cdot m$</td>
<td>Torque (3 x 1 Torque vector applied on each of body axis)</td>
<td>$N \cdot m$</td>
</tr>
</tbody>
</table>

**Table 3.11: Reaction wheels Simulink Block Inputs & Outputs**
- Nadir Pointing Mode
- Sun Pointing Mode

Depending on the value of the Mode that can be set by user at run time, a switch-case diagram implements the selection of any of the modes. The only selected block is executed and the merge block transmits the only selected data. In case of Sun Pointing and Nadir Pointing mode, the output of Flight Software is torques to be applied on the body in order to correct the attitude error. In case of Detumbling Mode, the output of flight software is the Dipole Commands generated for each of the three magnetorquer rods placed at each of body axis.

![Flight Software Top Level hierarchy in Simulink](image)

**Figure 3.13:** Flight Software Top Level hierarchy in Simulink

### 3.6.1 Detumbling Mode

When the satellite is separated from the launcher and is ejected into the orbit, the initial angular rates along the axes of body are unexpected and uncertain. In most cases, the magnitude of these angular rates is around 5 to 10 degrees per second. In order to start the normal mission life for the satellite, it is required to damp these rates under certain threshold. The
threshold is dependent on the mission analysts’ requirements and is decided such that it can allow the satellite to start its mission modes.

In order not to drain the battery power and to perform the detumbling in a power efficient way, the magnetic control is used. Only magnetometer and magnetorquer rods are working in this mode. The basic magnetic control law is called *b-dot bang bang law*. The theory of law is that the derivative of magnetic field intensity is calculated at each time stamp and the maximum torque is applied in the opposite direction to that. This law is efficient mostly in detumbling mode. The law can be written as:

\[ M_{\text{bdot}} = -M_{\text{max}} \cdot \text{sign}(\dot{B}_b) \] (3.10)

The maximum saturation dipole, that can be provided on each body axis is 6Am\(^2\). This value is chosen as the normal range of dipole for commercially available torquer rods for cubesats and small satellites [3].

There is another method of implementing detumble control, which is the *proportional gain bdot controller*. In this law, a constant positive gain is multiplied with the normalized vector of rate of change of the magnetic field. The *B-dot proportional control law* could be written as [3]:

\[ M_{\text{bdot}} = -k_b \cdot \frac{\dot{B}_b}{||B_b||} \] (3.11)

The value of the proportional gain \( k_b \) is chosen as 1024 [3]. The *B-dot Proportional Control law* is considered in this report for detumbling mode and the results of same are presented in the results chapter. For Simulink model development, both of these control laws are presented and one can be chosen using *switch*.

### 3.6.2 Sun Pointing Mode

In this mission mode, the satellite is made to point towards the sun so that solar panels may charge the batteries. This is a vital and basic mission mode which is necessary to achieve and maintain in every satellite.

This block relies on the quaternions error calculation which is part of the Cubesat Simulation library. There are two alignment vectors and two reference vectors are provided to the block. It tries to align the first reference vector with first alignment vector and then it tries to align the second
Attitude Control System

Figure 3.14: Detumbling Mode Matlab/Simulink Block

<table>
<thead>
<tr>
<th>Input</th>
<th>Unit</th>
<th>Output</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$B_{MGM}$ (3 × 1 Magnetic Field Intensity Vector from Magnetometer)</td>
<td>Tesla</td>
<td>DipoleCMD (3 × 1 Dipole Command vector in each of body axis)</td>
<td>$A-m^2$ (Ampere-square-meter)</td>
</tr>
<tr>
<td>$B_{BODY}$ (3 × 1 Magnetic Field Intensity Vector from Magnetometer)</td>
<td>Tesla</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.12: Detumbling Mode Simulink Block Inputs & Outputs

reference vector with second alignment vector. The first align vector is -z axis of satellite, which is required to face towards sun. Similarly, the first reference vector is the sun vector rotated by the attitude quaternions between ECI frame and Body frame.

$$FirstAlignmentVector = [0 \ 0 \ -1] \quad (3.12)$$

The second alignment vector is the x-axis of the satellite which is required to align with the z-axis of ECI frame, roughly called north pole of the earth.

$$SecondAlignmentVector = [1 \ 0 \ 0] \quad (3.13)$$
The error quaternions are calculated and then transformed into Euler angles. A PID controller takes the error Euler angles as input and tries to calculate the required torque commands. These commands are then directed to the actuators for further actions. The gains of PID controller are as follows:

\[ K_P = 1.00 \times 10^{-4} \]
\[ K_I = 2.00 \times 10^{-10} \]
\[ K_D = 0.02 \]

<table>
<thead>
<tr>
<th>Input</th>
<th>Unit</th>
<th>Output</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>xSunECI</td>
<td>m</td>
<td>TorqueCMD</td>
<td>N − m</td>
</tr>
<tr>
<td>(3 × 1 Sun position</td>
<td>meters</td>
<td>(3 × 1 Torque Command</td>
<td>(Newton-meter)</td>
</tr>
<tr>
<td>Vector in ECI frame)</td>
<td></td>
<td>vector in each body axis)</td>
<td></td>
</tr>
<tr>
<td>qECI2b</td>
<td>-</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(4 × 1 Quaternions</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>between Body frame</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>to ECI frame)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3.13: Sun Pointing Mode Simulink Block Inputs & Outputs
3.6.3 Nadir Pointing Mode

This mission mode tries to point the satellite face containing scientific equipment towards the earth, considering it is a LEO satellite with imaging payload. This mode relies on the position and velocity information in ECEF frame generated by GPS and the quaternions generated by dynamics block between body frame and ECEF frame.

Firstly, the quaternions between position and velocity is calculated which can be represented as quaternions between ECEF frame and ORF (Orbital Reference Frame). Then, the quaternions multiplication is performed in order to calculate the quaternions angle between body frame and ORF frame, which is the actually quaternion error. This is then transformed into Euler angles and a similar PID controller tries to minimize this error by generating the required torque for the actuators. The gains of PID controller are as follows:

\[
K_P = 1.00 \times 10^{-4} \\
K_I = 2.00 \times 10^{-10} \\
K_D = 0.02
\]

3.6.4 Visualization

Matlab Simulink provides an interface to visualize three-dimensional animation of the satellite orbit and attitude maneuvers. There is an already developed Library model for animation provided with Simulink CubeSat.
### Attitude Control System

<table>
<thead>
<tr>
<th>Input</th>
<th>Unit</th>
<th>Output</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>xECEF (3 × 1 Satellite Position Vector in ECEF frame)</td>
<td>m</td>
<td>TorqueCMD (3 × 1 Torque Command vector in each of body axis)</td>
<td>N – m (Newton–meter)</td>
</tr>
<tr>
<td>vECEF (3 × 1 Satellite Velocity Vector in ECEF frame)</td>
<td>m/s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>qECEF2b (4 × 1 Quaternions between Body frame to ECEF frame)</td>
<td>-</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 3.14**: Nadir Pointing Mode Simulink Block Inputs & Outputs

Simulation that takes the information of sun position vector in ECI frame, attitude of space craft with reference to ECEF and ECI frames, earth rotation parameters and space craft position in ECEF frame of reference. This block creates a 3D model to visualize the motion and maneuver of satellite updated at each time stamp. The mission modes and angular rates of body can be analyzed in great convenience through it. In the following figure, a screenshot of a frame of animation is presented.
Figure 3.17: Three-Dimensional Visualization of satellite in Orbit
Chapter 4

Radiations Hardened Software

To develop a sophisticated mathematical algorithm for controlling the attitude of the satellite and embed it into the hardware is not enough for the space application. The space craft is constantly exposed to the huge amount of radiations and charged particles, which may not only damage the physical structure of the satellite but instead some of them may penetrate into the devices and cause transient and permanent damage. The hardware and software of the space craft both are vulnerable to the radiations’ damage.

The developed mathematical model for attitude control, discussed in previous chapters, is made radiation hardened in order to tolerate the unexpected event of radiation hazards. Only the SEUs are considered and mitigated in this work and therefore the work methodology for same is discussed in this chapter.

4.1 Development of Flight Software

The part of the matlab/Simulink model that is supposed to be embed inside the hardware is the Flight Software for control algorithm. All the remaining parts of Simulink model are developed and simulated in order to provide the realistic scenario for the development and testing of Flight Software. The flight software is developed in the C++ language and tested in order to make sure that it provides the same required functionalities and then the radiation hardening technique is employed in order to make the system work properly
Radiations Hardened Software

...in the presence of radiations.

In this section, the development of flight software and its major components and hierarchy is discussed.

### 4.1.1 Data Structures

There are few global data structures defined to implement the flight software code. The purpose of global declaration of these data structures is convenient accessibility to data for every component of software by not passing the values or references between sub-routines.

The following data structures are defined in this study:

<table>
<thead>
<tr>
<th>Data Structure</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExtUVehicleT</td>
<td>External Inputs</td>
</tr>
<tr>
<td>ExtYVehicleT</td>
<td>External Outputs</td>
</tr>
<tr>
<td>ConstBVehicleT</td>
<td>Constants Data structure</td>
</tr>
<tr>
<td>DWVehicleT</td>
<td>States variables Data Structure</td>
</tr>
<tr>
<td>BVehicleT</td>
<td>Block Signals which are defined for intermediate operations on data</td>
</tr>
</tbody>
</table>

Table 4.1: List Of Data Structures in C++ Flight Software

### 4.1.2 Software Hierarchy

The software is divided into different subroutine files in order to better understand, debug and modify the code. The hierarchy can be adopted for any mission algorithm. List of important files is given:

- **FlightSoftware**: This function is called from the Simulink s-function as the entry point, which further calls the other functions.

- **Initialization**: It initializes all data structures with zeros or predefined constants.

- **Vehicle Step**: This function calls the relevant subroutine for mission modes depends on the mode selection using a *switch-case* scenario.
Radiations Hardened Software

- **Detumbling Mode**: This function implements the *Detumbling mode* for the satellite mission.

- **Nadir Pointing**: This function implements the *Nadir Pointing mode* for the satellite mission.

- **Sun Pointing**: This function implements the *Sun Pointing mode* for the satellite mission.

- **Vehicle.h** stores the definitions of all data structures to be used in the flight software.

### 4.2 Development of Radiation Hardened Flight Software

In the work presented here, only the flight software block of closed loop model is converted into C++ language software code, as this is the only part that has to be embedded into the hardware and face the radiation noises. This code is then analyzed in detail in order to employ the effect of SEUs and its mitigation techniques.

The development of radiation hardened software has two major parts:

- Study the effect of Single Event Upset on embedded software and model it accordingly on the flight software.

- Integrate the radiation hardening technique to make the flight software tolerant against radiation events.

#### 4.2.1 Radiation Effects on Software

Considering the software aspect, the radiations can corrupt the data stored in the memory devices by bit flipping. This effect is usually dealt in great precision by data scrubbers that constantly correct and update the corrupted data with the help of several error correction codes.

Radiations can also corrupt the running software by flipping some bits in the micro-controller when the software is running. This problem is unavoidable in some cases and may cause them serious errors. This can cause the code to jump to an unexpected memory locations or any unwanted routines may execute.
In the work presented in this report, the bit flipping event is simulated by calling a subroutine that randomly flips any bit of any of the running data structure. This method is a close approximation to what can happen during the SEU event during the flight. The probability of bit flipping event can be controlled from outside by the user. User can choose any number zero or greater than zero, and in result of that, the subroutine that causes SEU will be called that many times at each time stamp.

### 4.2.2 Radiation Hardening Library

There is a Radiation Hardening Library provided, written in C++ programming language, which is discussed in the chapter 1, that provides the interface to the user to declare a special object of any data type which maintains the triple redundancy on the level of variable. The example of variable declaration is shown:

\[
\text{HardenedData :: TripleData < Datatype > Variable; (4.1)}
\]

The three copies of data can be accessed as:

\[
\begin{align*}
\text{Variable.a();} \\
\text{Variable.b();} \\
\text{Variable.c();}
\end{align*}
\]

Reading of the variable gives the majority voted value and writing the variable updates the three copies of data. All the basic operators are overridden to maintain the seamless operations of data in \textit{TripleData.h} file.

There is also another file, \textit{SingleData.h} provided which implements the same functions, but it only keeps one copy of data. The purpose of this file is to not change the software code when choosing \textit{radiation hardened} or \textit{unhardened} software. The example of variable declaration is shown:

\[
\text{HardenedData :: SingleData < Datatype > Variable; (4.2)}
\]

The data can be accessed as:

\[
\text{Variable.a();}
\]
4.2.3 Data Structures

The same data structures are kept for radiation hardened software. In order to model the radiation effects, every data component is accessible globally to the noise injection sub-routine. The only difference is that all variables in data structures are declared as *TripleData* type and *SingleData* type. A macro variable `RAD-HARD` can be defined or undefined which can select the definitions of *Triple Data* or *Single Data*.

4.2.4 Software Hierarchy

The radiation hardened flight software has few hierarchical changes as compared to the normal software. Following subroutines are added in order to model and support the hardening effects.

- **Radiation Effects**: This function implements the SEU subroutine for the modeling of bit flip event in the flight software.

- **TripleData.h** implements the overridden functions of all operators for the *Triple Data* type variables.

- **SingleData.h** implements the overridden functions of all operators for the *Single Data* type variables.
4.2.5 Radiation Effects Sub-Routine

The subroutine that causes the Bit Flipping is called every cycle if the flag Radiation Noise Flag is set to 1 by user. This sub routine is able to corrupt the value of any element of any data structure. It is called just before the Vehicle Step so that the corrupted data will affect the most part of software code. This subroutine generates a random number at each call and selects a random data from any data structure and changes its values to the random number.

It is also possible to control the number of calls to this subroutine in each cycle by user by setting the external input variable Number of Bit Flip events. This will define the probability of Bit Flipping event. In order to corrupt the data, a copy of data is corrupted by accessing its one copy. In order to employ the smooth transition of definition between Triple Data and Single Data, the copy of data is corrupted like the following:

\[ \text{Variable.a()} = \text{RandomNumber}; \]

4.2.6 Radiation Tolerance

To fight against the effect of Radiations, the data structures are declared with each item in the way mentioned above as TripleData. In this way, the data corruption only occur to one of the copy of any variable while the other two copies remain valid. Although, the software code will take three times more space for data structures, but the whole flight software will be hardened against radiations effects. More number of bit flipping per cycle will cause the corruption of more variables in each cycle. But, the radiation hardened software will keep tolerating this effect.

4.2.7 S-functions Implementation

S-functions feature of the Matlab/Simulink is a gateway between C/C++ or VHDL code and Simulink model. In the work presented here, the flight software block of Simulink is converted into generic C++ code and the radiation hardening technique, presented above, is implemented and tested. Then an s-function is created using s-function builder in Simulink which calls the C++ function through Simulink. To the other blocks, it behaves like a black box and acts like a normal subsystem block that takes the same inputs as the Flight Software, but inside it runs the C++ code. User can select
whether to inject the radiation effects or not, and how much bit flip events per cycle can happen can also be controlled by sliding mode gain controller. When the *Radiation Noise Flag* is set to zero, the *Number of Bit Flip Events* do not have any effect.

**Figure 4.2:** S-function Builder Representation of Flight Software
Chapter 5

Simulations and Results

This chapter summarizes the results of simulations and mathematical modeling performed during development and testing of radiation hardened attitude control algorithm.

5.1 Matlab & Simulink Results

In this section, the simulation results of Attitude Control Algorithm without the presence of radiations and without any radiation hardening technique are presented.

5.1.1 Development Platforms & Tools

Matlab/Simulink platform is used for development and testing of closed loop simulation of attitude control algorithm. For the following results, the *s-function* is not the part of loop for the simulations.

5.1.2 Mission Modes

In order to better visualize the results of the simulations, it is better to keep the mission modes separated and analyze their outcomes.

Detumbling Mode

Initially, when the satellite is separated from the launcher, it is assumed that it has certain angular rates along its body, which is usually 5 to 7 degrees
Simulations and Results

per second. The normal missions are assumed to be detumbled when their rates have dropped below 0.5 degrees per second in each axis, but it depends on the control as well as the mass of the satellite. Due to the very basic control of rate of change of magnetic field, the detumble $b\dot{b}$ law is useful when the satellite has smaller angular rates. In the study presented in this report, the initial angular rates along the x-y-z body axes are following:

$$\omega_b = [5 \quad 4 \quad -0.0517] \quad \text{deg/sec}$$

(5.1)

The following is the simulation result achieved for Detumble mode. The angular rates estimation along the body axes is the observation of this mode. It can be seen in the figure that in the start when the angular rates are high, the control law tends to act stronger, but eventually it slows down the rates along the all body axes within 0.5$\text{deg/sec}$. The control law took almost 6 hours to damp the rates.

Figure 5.1: Detumbling Mode Simulation Results: Angular rates along xyz body axes

Nadir Pointing Mode

In this mode, the control algorithm is required to generate a set of torque commands in order to keep pointing the z-axis of satellite towards the center of earth. When the detumbling mode is succeeded in achieving the angular rates below a mission specific certain threshold, the nadir pointing mode can be switched to point the scientific equipment towards center of earth.
Simulations and Results

The angular rates at the start of this mode in the body are considered as $0.4 \text{deg/sec}$ in $x$ and $y$ axes, and $-0.0517 \text{deg/sec}$ in $z$ axis. This mode relies on the reaction wheels thus the required attitude is achieved relatively fast. The attitude error in the form of Euler angles can be seen in the figure that the pointing error is start to reduce very fast initially and under 1000 seconds, the attitude error is damped within the limits of 0.05 degrees which is equals to 180 arc-seconds. The zoom-in of the curve is also shown for 8000 seconds which represents the steady state error and shows that the attitude error remains under 0.05 degrees.

![Figure 5.2: Nadir Pointing Mode Attitude Error: Euler angles](image)

47
Sun Pointing Mode

In this mode, the control algorithm is required to generate a set of torque commands in order to keep pointing the -z-axis of satellite towards the sun. This mode can be worked when the satellite is out of the solar eclipse. Therefore, it is considered that initially during the eclipse, the Nadir Pointing Mode is working, and when the satellite is out of the shadow, the Sun Pointing Mode is switched. As this mode also relies on the reaction wheels, so the attitude error is reduced faster than detumbling mode.

In the figure, it can be seen that around 500 seconds after the simulation started, the mode is switched to Sun Pointing Mode, and then after 2500 seconds, the attitude error reduced within 0.01 degrees which is equals to 36 arc seconds. Then, the steady state response of the controller can be seen in the other figure that the error remains same throughout the simulation.

5.2 Radiation Effects

When the radiation noise injection flag is set from the Simulink, the SEU subroutine is called at each timestamp and a randomly selected data value is corrupted each cycle in order to visualize and simulate the exaggerated
Simulations and Results

Figure 5.4: Sun Pointing Mode Attitude Error: Euler angles

Figure 5.5: Sun Pointing Mode Attitude Error steady state: Euler angles

effect of radiations.
5.2.1 Mission Modes Simulation Results

The un-hardened software code will not be able to withstand the bit flipping of SEU. The results can be seen in the simulations.

Detumbling Mode

Detumbling mode will not be able to detumble the satellite towards threshold rates due to the radiation noises. The data structures employ the *Single Data* header file for data elements declaration and the *Radiation Noise Flag* is set to 1 in order to inject noises. *Number of Bit Flip Events* is also set to 1 which is enough to corrupt the software. As the bit flipping event simulation causes the random number assignment to the victim data, therefore, the closed loop simulation will not be able to proceed with the huge values of $B_{\text{Body}}$ and *Commands*. Therefore, it is necessary to put a saturation block on the output of s-function block. The general trend shows that the maximum values of $B_{\text{Body}}$ are in the range of $-5 \times 10^{-5}$ to $5 \times 10^{-5}$, and the maximum values of *Commands* can vary from -10 to 10. The saturation block limits the output of s-function block under these values.

In the following figure, the detumbling mode simulation is shown which represents that the flight software is unable to detumble the body angular rates due to the constant bit flipping events.

![Figure 5.6: Radiations affected Detumbling Mode Simulation Results: Angular rates along xyz body axes](image-url)
Nadir Pointing Mode

Nadir Pointing mode accuracy and efficiency can be seen as the attitude error after the noise effects as in the following figure. The attitude error should ideally reduce to minimum values after few seconds but due to the bit flipping events every cycle, the attitude error changes every second. Therefore, unwanted torques will be produced every second and the required attitude will not be achieved. In the following figure, when the \textit{Radiation Noise Flag} is set to 1, the attitude error constantly changes to different values. Therefore, it is impossible to achieve the required attitude.

![Figure 5.7: Radiations affected Nadir Pointing Mode Simulation Results: Euler angles](image)

Sun Pointing Mode

Sun Pointing mode accuracy and efficiency can be seen as the attitude error after the noise effects as in the following figure. When the satellite is out of the eclipse, the mode is switched to Sun Pointing Mode and then \textit{Radiation Noise Flag} is set to 1 for the bit flipping simulation. Due to the random bit flipping, the attitude error constantly changes and the unwanted torques are applied on the body. The satellite may experience unwanted abrupt huge torque commands to actuators and the desired attitude will be impossible to achieve.
5.3 Radiation Hardened Software

After implementing the radiation hardening technique in software, this version of control algorithm should be able to withstand the radiation noises. The SEU effects should be mitigated by the triple data redundancy technique.

5.3.1 Mission Modes Simulation Results

The hardened control algorithm for each mode can be analyzed by simulation results.
Detumbling Mode

The detumbling mode should be able to work normally and the rates along the body axes should tend to detumble even in the presence of radiation noises, as seen in figure. The initial angular rates along body were chosen same as previous detumbling mode. The Radiation Noise Flag is set to 1 externally and Number of Bit Flip Events is set to 2, which could be any value. The system withstand the radiation effects and preformed the similar way as normal detumbling mode performed in the absence of radiation effects.

Figure 5.9: Radiation Hardened Detumbling Mode Simulation Results: Angular rates along xyz body axes

Nadir Pointing Mode

Nadir Pointing mode should work seamlessly as it was working without radiation noises. The hardening technique should tolerate the SEU effects and keep minimizing the attitude error effectively, as seen in figure. The initial conditions for this simulation is chosen as the previous. In the initial 1000 seconds, the attitude error reduced under 0.07 degrees which is equals to 252 arc seconds and for the rest of the simulation, the steady state error remains under 0.07 degrees.
Sun Pointing Mode

Sun Pointing mode should work seamlessly as it was working without radiation noises. The hardening technique should tolerate the SEU effects and keep minimizing the attitude error effectively, as seen in figure. The Sun Pointing mode is switched as soon as the satellite came out of the solar eclipse.

After 1000 seconds, the mode is switched to Sun Pointing, and the attitude error is reduced under 0.035 degrees, which is 126 arc-seconds, after 3000 seconds. The attitude error remains same for the rest of simulation.
5.4 Comparative Analysis

It can be seen from the results that the designed attitude control system, when working in the absence of radiations, achieved the required results. But, due to the SEU effects, the software affected badly and it could not proceed with the mission objectives in any of the mode. Then, the radiation hardening technique again achieved the required performance even in the presence of radiations.

The performance parameters comparison for each of the mission mode are summarized in the following tables.
Figure 5.12: Radiation Hardened Sun Pointing Mode Simulation Results: Euler angles

<table>
<thead>
<tr>
<th>Detumbling Mode</th>
<th>Damped Rates</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulink Model</td>
<td>0.5 deg/sec</td>
<td>It took 6 hours to damp the rates within threshold</td>
</tr>
<tr>
<td>Radiations Affected s-function</td>
<td><em>Un-damped</em></td>
<td>The angular rates of satellite remains constant around 3 deg/sec even after 6 hours</td>
</tr>
<tr>
<td>Radiations Hardened s-function</td>
<td>0.5 deg/sec</td>
<td>After 6 hours, the rates are damped within threshold</td>
</tr>
</tbody>
</table>

Table 5.1: Detumbling mode performance comparison
Simulations and Results

Figure 5.13: Radiation Hardened Sun Pointing Mode steady state Simulation Results: Euler angles

<table>
<thead>
<tr>
<th>Nadir Pointing Mode</th>
<th>Pointing Accuracy</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulink Model</td>
<td>180 arc-sec</td>
<td>After 1000 seconds, the attitude error remains under threshold.</td>
</tr>
<tr>
<td>Radiations Affected s-function</td>
<td>Failed to point</td>
<td>The attitude error constantly changes due to radiations and unwanted torques on the body make it impossible to achieve the desired attitude.</td>
</tr>
<tr>
<td>Radiations Hardened s-function</td>
<td>252 arc-sec</td>
<td>After 1000 seconds, the attitude error remains under threshold.</td>
</tr>
</tbody>
</table>

Table 5.2: Nadir Pointing mode performance comparison
### Sun Pointing Mode Performance Comparison

<table>
<thead>
<tr>
<th>Sun Pointing Mode</th>
<th>Pointing Accuracy</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulink Model</td>
<td>36 arc-sec</td>
<td>After 2500 seconds, the attitude error remains under threshold.</td>
</tr>
<tr>
<td>Radiations Affected s-function</td>
<td>Failed to point</td>
<td>The attitude error constantly changes due to radiations and unwanted torques on the body make it impossible to achieve the desired attitude.</td>
</tr>
<tr>
<td>Radiations Hardened s-function</td>
<td>126 arc-sec</td>
<td>After 3000 seconds, the attitude error remains under threshold.</td>
</tr>
</tbody>
</table>

**Table 5.3:** Sun Pointing mode performance comparison
Chapter 6

Conclusion

In the work presented in this report, a basic attitude control system of a small satellite for low earth orbit mission is designed and simulated. *Detumbling Mode, Nadir Pointing Mode* and *Sun Pointing Mode* are developed and modeled using Matlab/Simulink. The closed loop mathematical model developed on Simulink achieves these basic functional modes of satellite. The *detumbling mode* damps the satellite angular body rates along x, y and z axes under 0.5 deg/sec. The magnetic control of this mode much longer time, but eventually settles down the body rates. The *Sun Pointing mode* and *Nadir Pointing mode* implements 3-axis stabilization and provides the pointing accuracy of 36 arc-sec and 180 arc-sec respectively.

The effects of radiations are considered that can affect the running software of the satellite, more precisely, the effects of Single Event Upset that can cause the bit-flipping in the software is considered and modeled. In order to analyze the effects of SEU, the control algorithm which is also called the flight software, is converted into C++ language code and the SEUs are modeled on that software. This software is plugged back into the Simulink model using *s-function* technique so that the closed loop simulation can analyze the performance of software in the presence of radiations. Upon the simulation of radiation effects, it is concluded that the random bit flipping of the software may lead the code flow into an unexpected state or can damage the data so badly that the several calculations of corrupted data can unstable the whole system with the failure of achieving mission objectives.

The software is then made hardened against the radiations using a software technique which triplicates the data. The radiation-hardened software runs in the closed loop Simulink model to prove that the software achieves the
required functionality and performance. The hardened software makes the system tolerable with the SEU events. The detumbling mode settles down the body angular rates along x, y and z axes of satellite under 0.5 deg/sec while the Sun Pointing mode and Nadir Pointing mode provides the pointing accuracy of 126 arc-sec and 252 arc-sec respectively, which are acceptable.

6.1 Future Work

The scope of flight software can be extended. The sensors’ raw data may be taken as an input in the form of currents and voltages and then processed to determine the required vectors and matrices. Similarly, the flight software outputs should be converted into electrical stimuli which are voltage or current signals and given to the actuators.

The reaction wheel block should have a desaturation block in order to lower down the speeds of reaction wheels without affecting the momentum of satellite. The proper commands for magnetorquer rods can be calculated in order to implement desaturation function.

The disturbances acting on the satellite are not considered in this work. They can be modeled to make the system behave even more realistic.

The radiation effects modeling can be extended. The SEU event should randomly happen during the execution of the software. The radiation effects subroutine can be called by a software interrupt randomly during the execution of any function in order to more randomize the event probability.

The development of code of flight software can be optimized by reusing many variables. But this might lead to increase in the probability of using the already corrupted data. On the other hand, defining long data structures and breaking down the operations into smaller operations increase the data size. The trade-off can be carefully chosen by analyzing the behaviour of system deeply.
Appendix A

Flight Software Source Code

FlightSoftware.cpp

```c++
#include "Vehicle.h"

/* The following is the declarations of data structures which are used in the flight software */
B_Vehicle_T Vehicle_B;

/* Block signals: The local variables of the software to store temporary values and intermediate calculations */

/* Block states: The state variables, that are required to store the values between different cycles */

/* External inputs: The inputs of the flight software */

/* External outputs: The outputs of the flight software */

/* Constants: These values are initialized in the start to store the constants */

constB_Vehicle_T Vehicle_ConstB;

void FlightSoftware(const double B_mgm[3], const double q_eci2b[4], const double xSunECI[3], const double qECEF2b[4], const double vECEF[3],
const double xECEF[3], const double Mode, double Commands[3], double Bbody[3], double AttitudeError[3],
unsigned int RadiationNoiseFlag, unsigned int NumberOfBitFlipEvents, double AttitudeError[3])
{
    static int IdxState = 0;
    unsigned int index = 0;
    if (IdxState == 0) {
        Vehicle_initialize();
        IdxState = 1;
    }

    void FlightSoftware(const double B_mgm[3], const double q_eci2b[4], const double xSunECI[3],
    const double qECEF2b[4], const double vECEF[3], const double xECEF[3],
    const double Mode, double Commands[3], double Bbody[3], unsigned int RadiationNoiseFlag, unsigned int NumberOfBitFlipEvents, double AttitudeError[3])
{
    static int IdxState = 0;
    unsigned int index = 0;
    if (IdxState == 0) {
        Vehicle_initialize();
        IdxState = 1;
    }

    /* Reading external inputs */
    Vehicle_U.Bmgm[0] = B_mgm[0];
    Vehicle_U.Bmgm[1] = B_mgm[1];
    Vehicle_U.q_eci2b[0] = q_eci2b[0];
    Vehicle_U.q_eci2b[1] = q_eci2b[1];
    Vehicle_U.q_eci2b[2] = q_eci2b[2];
    Vehicle_U.q_eci2b[3] = q_eci2b[3];
    Vehicle_U.xSunECI[0] = xSunECI[0];
    Vehicle_U.xSunECI[1] = xSunECI[1];
    Vehicle_U.qECEF2b[0] = qECEF2b[0];
    Vehicle_U.qECEF2b[1] = qECEF2b[1];
    Vehicle_U.qECEF2b[2] = qECEF2b[2];
    Vehicle_U.qECEF2b[3] = qECEF2b[3];
    Vehicle_U.vECEF[0] = vECEF[0];
    Vehicle_U.vECEF[1] = vECEF[1];
    Vehicle_U.xECEF[0] = xECEF[0];
    Vehicle_U.xECEF[1] = xECEF[1];
    Vehicle_U.xECEF[3] = xECEF[3];
    Vehicle_U.Mode = Mode;

    /* Single Event Upset Function */
    if (RadiationNoiseFlag == 1) {
        for (index = 0; index < NumberOfBitFlipEvents; index++)
            SEU(); // SEU() will be called the number of times defined by NumberOfBitFlipEvents
    }

    /* Calling main function */
    Vehicle_step();
```
void Detumble()
{
    real_T tmpForInput_idx_0;
    /* Outputs for IAction SubSystem '<S1>/Detumble Mode' incorporates:
       * ActionPort: '<S2>/Action Port'
     */
    /* Output: '<Root>/Bbody' incorporates:
        * Import: '<Root>/Bmgm'
        * Import: '<S2>/Bmgm'
     */
    /* MATLAB Function 'Vehicle Flight Software/Detumble Mode/Derivative Function': '<S5>:1' */
    /* <S5>:1:1.4' if isempty (state), */
    /* <S5>:1:1.7' bdot = b-state, */
    /* <S5>:1:1.8' state = b, */
    Vehicle_Y.Bbody[0] = Vehicle_U.Bmgm[0];
    /* MATLAB Function: '<S5>/Derivative Function' incorporates:
       * Output: '<Root>/Bbody'
     */
    Vehicle_B.bdot[0] = Vehicle_Y.Bbody[0] - Vehicle_DW.state[0];
    Vehicle_DW.state[0] = Vehicle_Y.Bbody[0];
    /* Output: '<Root>/Bbody' incorporates:
        * Import: '<Root>/Bmgm'
        * Import: '<S2>/Bmgm'
     */
    /* MATLAB Function: '<S5>/Derivative Function' incorporates:
       * Output: '<Root>/Bbody'
     */
    Vehicle_DW.state[1] = Vehicle_Y.Bbody[1];
    /* Output: '<Root>/Bbody' incorporates:
        * Import: '<Root>/Bmgm'
        * Import: '<S2>/Bmgm'
     */
    /* MATLAB Function: '<S5>/Derivative Function' incorporates:
       * Output: '<Root>/Bbody'
     */
    /* MATLAB Function: '<S5>/Magnetroquer control' */
    /* MATLAB Function 'Vehicle Flight Software/Detumble Mode/Magnetroquer control': '<S6>:1' */
    /* <S6>:1:1.3' mmax=6, */
    /* <S6>:1:1.6' Dipole_x=[-mmax*sign(bdot(1));0;0]; */
    tmpForInput_idx_0 = (real_T)Vehicle_B.bdot[0];
    /* if (tmpForInput_idx_0 < 0.0) { */
    /* tmpForInput_idx_0 = -1.0; */
    /* } else if (tmpForInput_idx_0 > 0.0) { */
    /* tmpForInput_idx_0 = 1.0; */
    /* } else if (tmpForInput_idx_0 == 0.0) { */
    /* tmpForInput_idx_0 = 0.0; */
    /* } */
    Vehicle_B.norm_b = sqrt((Vehicle_Y.Bbody[0]*Vehicle_Y.Bbody[0])+(Vehicle_Y.Bbody[1]*Vehicle_Y.Bbody[1])+(Vehicle_Y.Bbody[2]*Vehicle_Y.Bbody[2]));
    if (Vehicle_B.norm_b == 0.0)
        Vehicle_B.norm_b = 1.7320508075688774e-09;
    /*Vehicle_B.Dipole_x[0] = -6.0 + tmpForInput_idx_0;*/
    Vehicle_B.Dipole_x[0] = -1024.0 * Vehicle_B.bdot[0]/Vehicle_B.norm_b;
    Vehicle_B.Dipole_x[1] = 0.0;
    Vehicle_B.Dipole_x[2] = 0.0;
    /* <S6>:1:1.7' Dipole_y=[0;-mmax*sign(bdot(2));0]; */
    Vehicle_B.Dipole_y[0] = 0.0;
    /* tmpForInput_idx_0 = (real_T)Vehicle_B.bdot[1];
        if (tmpForInput_idx_0 < 0.0) { */
    /* tmpForInput_idx_0 = -1.0; */
    /* } else if (tmpForInput_idx_0 > 0.0) { */
    /* tmpForInput_idx_0 = 1.0; */
    /* } else if (tmpForInput_idx_0 == 0.0) { */
    /* tmpForInput_idx_0 = 0.0; */
    /* } */
    /* Vehicle_B.Dipole_y[0] = -6.0 + tmpForInput_idx_0;*/
    Vehicle_B.Dipole_y[1] = -1024.0 * Vehicle_B.bdot[1]/Vehicle_B.norm_b;
    Vehicle_B.Dipole_y[2] = 0.0;
    /* <S6>:1:1.8' Dipole_z=[0;0;-mmax*sign(bdot(3));0]; */
NadirPointing.cpp

#include "Vehicle.h"

void NadirPointing() {
    real_T tmpForInput_idx_0;
    real_T tmpForInput_idx_1;
    real_T tmpForInput_idx_2;
    int32_T i;
    /* Outputs for If Action SubSystem: '<S3>/NadirPointing Mode' incorporates:
     + ActionPort: '<S3>/Action Port'
     */
    /* Product: '<S20>/j x k' incorporates:
    + Import: '<Root>/vECEF'
    + Import: '<Root>/xECEF'
    */
    /* Product: '<S20>/k x i' incorporates:
    + Import: '<Root>/vECEF'
    + Import: '<Root>/xECEF'
    */
    /* Product: '<S21>/i x j' incorporates:
    + Import: '<Root>/vECEF'
    + Import: '<Root>/xECEF'
    */
    Vehicle_B.i x j = Vehicle_U.vECEF[0] * Vehicle_U.xECEF[1];
    /* Product: '<S21>/k x i' incorporates:
    + Import: '<Root>/vECEF'
    + Import: '<Root>/xECEF'
    */
    Vehicle_B.k x i = Vehicle_U.vECEF[0] * Vehicle_U.xECEF[2];
    /* Product: '<S21>/j x k' incorporates:
    + Import: '<Root>/vECEF'
    + Import: '<Root>/xECEF'
    */
    Vehicle_B.j x k = Vehicle_U.vECEF[1] * Vehicle_U.xECEF[0];
    /* Sum: '<S13>/Sum' */
    Vehicle_B.Sum[0] = Vehicle_B.jxk - Vehicle_B.kxj;
    Vehicle_B.Sum[1] = Vehicle_B.kxi - Vehicle_B.i x k;
    Vehicle_B.Sum[2] = Vehicle_B.i x j - Vehicle_B.j x i;
    /* Gain: '<S7>/z = r' incorporates:
    + Import: '<Root>/xECEF'
    */
    Vehicle_B.z [0] = Vehicle_U.xECEF[0];
    Vehicle_B.z [1] = Vehicle_U.xECEF[1];
    /* Product: '<S18>/i x j' */
    Vehicle_B.i x j_f = Vehicle_B.Sum[0] + Vehicle_B.z [1];
    /* Product: '<S18>/j x k' */
    Vehicle_B.j x k_o = Vehicle_B.Sum[1] + Vehicle_B.z [2];
    /* Product: '<S18>/k x i' */
    Vehicle_B.k x i_c = Vehicle_B.Sum[2] + Vehicle_B.z [0];
    /* Product: '<S19>/i x k' */
    Vehicle_B.i x k_g = Vehicle_B.Sum[0] + Vehicle_B.z [2];
}
Vehicle_B_jxi_1 = Vehicle_B.Sum[1] + Vehicle_B.ar[0];

Vehicle_B_kxi_a = Vehicle_B.Sum[2] + Vehicle_B.ar[1];

Vehicle_B.Sum_l[0] = Vehicle_B.jxk_o - Vehicle_B.kxi_a;
Vehicle_B.Sum_l[1] = Vehicle_B.kxi_e - Vehicle_B.idx_g;

tmpForInput_idx_1 = Vehicle_B.Sum_l[0];
tmpForInput_idx_0 = Vehicle_B.Sum_l[0];
tmpForInput_idx_2 = tmpForInput_idx_1 * tmpForInput_idx_0;

tmpForInput_idx_1 = Vehicle_B.Sum_l[1];
tmpForInput_idx_0 = Vehicle_B.Sum_l[1];
tmpForInput_idx_2 += tmpForInput_idx_1 * tmpForInput_idx_0;

tmpForInput_idx_1 = Vehicle_B.Sum_l[2];
tmpForInput_idx_0 = Vehicle_B.Sum_l[2];
tmpForInput_idx_2 += tmpForInput_idx_1 * tmpForInput_idx_0;

tmpForInput_idx_2 = tmpForInput_idx_1 + tmpForInput_idx_0;

Vehicle_B.DotProduct = tmpForInput_idx_2;

Vehicle_B.SumofElements = tmpForInput_idx_0;

Vehicle_B.MathFunction1_b = std::sqrt(std::abs(tmpForInput_idx_0));

Vehicle_B.MathFunction1_c = std::sqrt(tmpForInput_idx_0);

Vehicle_B.ORFtoECEF[0] = Vehicle_B.Sum_l[0] / Vehicle_B.MathFunction1;

Vehicle_B.DotProduct_p = tmpForInput_idx_2;

Vehicle_B.SumofElements_c = tmpForInput_idx_0;

if (tmpForInput_idx_0 < 0.0) {
    Vehicle_B.MathFunction1_b = std::sqrt(std::abs(tmpForInput_idx_0));
} else {
    Vehicle_B.MathFunction1_c = std::sqrt(tmpForInput_idx_0);
}
```
/* About '<S16>/Math Function1' */
Vehicle_B.MathFunction1_b = std::sqrt(tmpForInput_idx_0);
}

/* Product: '<S16>/Divide' */

/* DotProduct: '<S17>/Dot Product */
tmpForInput_idx_1 = Vehicle_B.zr[0];
tmpForInput_idx_0 = Vehicle_B.zr[0];
tmpForInput_idx_2 = tmpForInput_idx_1 + tmpForInput_idx_0;

/* Product: '<S16>/Divide' */

/* DotProduct: '<S17>/Dot Product */
tmpForInput_idx_1 = Vehicle_B.zr[1];
tmpForInput_idx_0 = Vehicle_B.zr[1];
tmpForInput_idx_2 = tmpForInput_idx_1 + tmpForInput_idx_0;

/* Product: '<S16>/Divide' */

/* DotProduct: '<S17>/Dot Product */
tmpForInput_idx_1 = Vehicle_B.zr[2];
tmpForInput_idx_0 = Vehicle_B.zr[2];
tmpForInput_idx_2 = tmpForInput_idx_1 + tmpForInput_idx_0;

/* Product: '<S16>/Divide' */

/* DotProduct: '<S17>/Dot Product */
tmpForInput_idx_1 = Vehicle_B.zr[3];
tmpForInput_idx_0 = Vehicle_B.zr[3];
tmpForInput_idx_2 = tmpForInput_idx_1 + tmpForInput_idx_0;

/* Product: '<S16>/Divide' */

/* DotProduct: '<S17>/Dot Product */
tmpForInput_idx_1 = Vehicle_B.zr[4];
tmpForInput_idx_0 = Vehicle_B.zr[4];
tmpForInput_idx_2 = tmpForInput_idx_1 + tmpForInput_idx_0;

/* Product: '<S16>/Divide' */

/* DotProduct: '<S17>/Dot Product */
tmpForInput_idx_1 = Vehicle_B.zr[5];
tmpForInput_idx_0 = Vehicle_B.zr[5];
tmpForInput_idx_2 = tmpForInput_idx_1 + tmpForInput_idx_0;

/* Product: '<S16>/Divide' */

/* DotProduct: '<S17>/Dot Product */
tmpForInput_idx_1 = Vehicle_B.zr[6];
tmpForInput_idx_0 = Vehicle_B.zr[6];
tmpForInput_idx_2 = tmpForInput_idx_1 + tmpForInput_idx_0;

/* Product: '<S16>/Divide' */

/* DotProduct: '<S17>/Dot Product */
tmpForInput_idx_1 = Vehicle_B.zr[7];
tmpForInput_idx_0 = Vehicle_B.zr[7];
tmpForInput_idx_2 = tmpForInput_idx_1 + tmpForInput_idx_0;

/* Product: '<S16>/Divide' */

/* DotProduct: '<S17>/Dot Product */
tmpForInput_idx_1 = Vehicle_B.zr[8];
tmpForInput_idx_0 = Vehicle_B.zr[8];
tmpForInput_idx_2 = tmpForInput_idx_1 + tmpForInput_idx_0;

/* Product: '<S16>/Divide' */

/* DotProduct: '<S17>/Dot Product */
tmpForInput_idx_1 = Vehicle_B.zr[9];
tmpForInput_idx_0 = Vehicle_B.zr[9];
tmpForInput_idx_2 = tmpForInput_idx_1 + tmpForInput_idx_0;

/* Product: '<S16>/Divide' */

/* DotProduct: '<S17>/Dot Product */
tmpForInput_idx_1 = Vehicle_B.zr[10];
tmpForInput_idx_0 = Vehicle_B.zr[10];
tmpForInput_idx_2 = tmpForInput_idx_1 + tmpForInput_idx_0;

/* Product: '<S16>/Divide' */

/* DotProduct: '<S17>/Dot Product */
tmpForInput_idx_1 = Vehicle_B.zr[11];
tmpForInput_idx_0 = Vehicle_B.zr[11];
tmpForInput_idx_2 = tmpForInput_idx_1 + tmpForInput_idx_0;

/* Product: '<S16>/Divide' */
```

382 Vehicle_B . Switch_b [0] = 0.5 ;
381
380
379 /∗
378 i f ( Vehicle_B . sqrt_l != 0.0 ) {
377 Vehicle_B . Add_ek = Vehicle_B .ECEFtoORF[3];
373
372 Vehicle_B . sqrt_l = std::sqrt( Vehicle_B . Add_f) ;
371 Vehicle_B .ECEFtoORF[4]) + 1.0 ;
370
369
368 /∗
367 i f ( Vehicle_B .ECEFtoORF[8] > Vehicle_B .ECEFtoORF[0] ) {
366 Vehicle_B . Merge_i[0] = Vehicle_B . Product_o[2];
364 Vehicle_B . Merge_i[1] = Vehicle_B . Product_o[0];
361 Vehicle_B . Product_o[0] = Vehicle_B . Add_gw;
360
359
358 /∗
357 i f ( Vehicle_B . sqrt_mw != 0.0 ) {
355
354 /∗
352 i f ( ( Vehicle_B .ECEFtoORF[4] > Vehicle_B .ECEFtoORF[0] ) &&
347
346
345
344 Vehicle_B . Add_a = Vehicle_B .ECEFtoORF[3];
342 Vehicle_B . Add_k = Vehicle_B .ECEFtoORF[7];
341 Vehicle_B . Gain1 = 2.0;
340 Vehicle_B . Merge_i[0] = 0.5;
339
338
337 /∗
336 Vehicle_B . sqrt_mw ;
335 Vehicle_B . sqrt_l ;
334
333
332
331 i f ( Vehicle_B . sqrt_l != 0.0 ) {
329
328 /∗
327 i f ( Vehicle_B .ECEFtoORF[4] > Vehicle_B .ECEFtoORF[0] ) {
323
322
321
320 Vehicle_B . sqrt_l ;
319
318
317
316
315
314
313
312
311
310 /∗
309 i f ( Vehicle_B .ECEFtoORF[4] > Vehicle_B .ECEFtoORF[0] ) {
308 Vehicle_B . Merge_i[0] = Vehicle_B . Add_d / Vehicle_B . Gain1;
305
304 i f ( ( Vehicle_B .ECEFtoORF[4] > Vehicle_B .ECEFtoORF[0] ) &&
300
303
302
301
300 /∗
299 i f ( ( Vehicle_B .ECEFtoORF[4] > Vehicle_B .ECEFtoORF[0] ) &&
297 Vehicle_B . Merge_i[0] = Vehicle_B . Add_d / Vehicle_B . Gain1;
294
293
292
291
290 Vehicle_B . sqrt_mw ;
289 Vehicle_B . sqrt_l ;
288
287
286 Vehicle_B . sqrt_l ;
285
284
283 Vehicle_B . Gain1 = 2.0;
282
281
280 Vehicle_B . sqrt_l ;
279
278
277
276
275 /∗
274 i f ( Vehicle_B .ECEFtoORF[4] > Vehicle_B .ECEFtoORF[0] ) {
272
271 /∗
270 i f ( Vehicle_B .ECEFtoORF[4] > Vehicle_B .ECEFtoORF[0] ) {
268
267 /∗
266 i f ( Vehicle_B .ECEFtoORF[4] > Vehicle_B .ECEFtoORF[0] ) {
264
263 /∗
262 i f ( Vehicle_B .ECEFtoORF[4] > Vehicle_B .ECEFtoORF[0] ) {
260
259 /∗
258 i f ( Vehicle_B .ECEFtoORF[4] > Vehicle_B .ECEFtoORF[0] ) {
256
255 /∗
254 i f ( Vehicle_B .ECEFtoORF[4] > Vehicle_B .ECEFtoORF[0] ) {
252
251 /∗
250 i f ( Vehicle_B .ECEFtoORF[4] > Vehicle_B .ECEFtoORF[0] ) {
248
247 /∗
246 i f ( Vehicle_B .ECEFtoORF[4] > Vehicle_B .ECEFtoORF[0] ) {
244
243 /∗
242 i f ( Vehicle_B .ECEFtoORF[4] > Vehicle_B .ECEFtoORF[0] ) {
240
239 /∗
238 i f ( Vehicle_B .ECEFtoORF[4] > Vehicle_B .ECEFtoORF[0] ) {
Vehicle_B.Product_d = Vehicle_B.Product_cy[2];
Vehicle_B.Merge_i[3] = Vehicle_B.Product_cy[1];
Vehicle_B.Merge_i[2] = Vehicle_B.Product_cy[0];
Vehicle_B.Product_cy[0] = Vehicle_B.Product_k;

Vehicle_B.Add_l = Vehicle_B.ECEFtoORF[7];
Vehicle_B.Add_o = Vehicle_B.ECEFtoORF[1] + Vehicle_B.ECEFtoORF[3];
Vehicle_B.Product_d = Vehicle_B.Switch[0] / Vehicle_B.Switch[1];
Vehicle_B.Product_d = Vehicle_B.Switch[1] / Vehicle_B.Switch[0];

Vehicle_B.Product_k = Vehicle_B.Switch[0] / Vehicle_B.Switch[1];
Vehicle_B.Add_o = Vehicle_B.ECEFtoORF[1] + Vehicle_B.ECEFtoORF[3];
Vehicle_B.Add_o = Vehicle_B.ECEFtoORF[1] + Vehicle_B.ECEFtoORF[3];
Vehicle_B.Product_d = Vehicle_B.Switch[0] / Vehicle_B.Switch[1];
Vehicle_B.Product_d = Vehicle_B.Switch[1] / Vehicle_B.Switch[0];
Vehicle_B.Tsamp[0] = Vehicle_B.DerivativeGain[0] 


// Gain: '<S86>/Derivative Gain' */ 
Vehicle_B.DerivativeGain[0] = 0.02 * Vehicle_B.VectorConcatenate_a[0]; 

// SampleTimeMath: '<S88>/Tsamp' 
* 
* About '<S88>/Tsamp': 
* y = u * K where K = 1 / ( w * Tsamp ) 
* Multiplication by K = weightedTsampQuantized is being 
* done implicitly by changing the scaling of the input signal. 
* No work needs to be done here. Downstream blocks may need 
* to do work to handle the scaling of the output; this happens 
* automatically. 
* 
* Vehicle_B.Tsamp[0] = Vehicle_B.DerivativeGain[0]; 

// Delay: '<S86>/UD' */

#include "Vehicle.h"

void SunPointing()
{
    real_T tmpForInput;
    real_T tmpForInput_0;
    real_T tmpForInput_1;
    real_T tmpForInput_idx_0;
    real_T tmpForInput_idx_1;
    real_T tmpForInput_idx_2;

    if (Vehicle_ConstB.Abs < 1.0E-6) {
        // Vehicle_IfActionSubsystem(&Vehicle_B.Merge_m);
        Vehicle_B.Merge_m = 1.0;
    } else if (Vehicle_ConstB.Abs1 < 1.0E-6) {
        // Vehicle_IfActionSubsystem1(&Vehicle_B.Merge_m);
        Vehicle_B.Merge_m = -1.0;
    } else {
        // Vehicle_IfActionSubsystem2(&Vehicle_B.Merge_m);
        Vehicle_B.Merge_m = 0.0;
    }

    // Product: '<S259>/Product' incorporates:
    * Import: '<Root>/q_eci2b';
    * Vehicle_B.Product_n = Vehicle_U.q_eci2b[0] * Vehicle_U.q_eci2b[0];
    * Vehicle_B.Sum_j = ( (Vehicle_B.Product_n + Vehicle_B.Product1_ii) + Vehicle_B.Product2_c ) + Vehicle_B.Product3_d;
    * Vehicle_B.sqrt_n = std::sqrt( Vehicle_B.Sum_j);
    * Vehicle_B.Product6 = Vehicle_B.Product2_oi * Vehicle_B.Product2_oi;
    * Vehicle_B.Product7 = Vehicle_B.Product3_a * Vehicle_B.Product3_a;
    * Vehicle_B.Product7 = Vehicle_B.Product3_a * Vehicle_B.Product3_a;
    * Vehicle_B.Sum3 = (0.5 - Vehicle_B.Product6) * Vehicle_B.Product7;
    * Vehicle_B.Gain2 = 2.0 * Vehicle_B.Sum3;
    * Product: '<S255>/Product8' incorporates:
}
/* Sum: '<S256>/Sum' */
Vehicle_B.Sum_lm = (Vehicle_B.Product4_k + Vehicle_B.Product8_c) +
                  Vehicle_B.Product5_l;
/* Product: '<S257>/Product' */
Vehicle_B.Product_c3 = Vehicle_B.Product1_fv + Vehicle_B.Product3_a;
/* Product: '<S257>/Product1' */
Vehicle_B.Product1_c = Vehicle_B.Product cw + Vehicle_B.Product2_oi;
/* Sum: '<S257>/Sum1' */
Vehicle_B.Sum_1_bu = Vehicle_B.Product_c3 + Vehicle_B.Product1_c;
/* Gain: '<S257>/Gain' */
Vehicle_B.Gain_k = 2.0 * Vehicle.B.Sum1_bu;
/* Product: '<S257>/Product4' incorporates:
  * Import: '<Root>/xSunECI' */
Vehicle_B.Product4_m = Vehicle_U.xSunECI[0] + Vehicle_B.Gain_k;
/* Product: '<S257>/Product2' */
Vehicle_B.Product2_k = Vehicle_B.Product cw + Vehicle_B.Product1_fv;
/* Product: '<S257>/Product3' */
Vehicle_B.Product3_f = Vehicle_B.Product2_oi + Vehicle_B.Product3_a;
/* Sum: '<S257>/Sum2' */
Vehicle_B.Sum_2_p = Vehicle_B.Product3_f - Vehicle_B.Product2_k;
/* Gain: '<S257>/Gain1' */
Vehicle_B.Gain1_j = 2.0 * Vehicle.B.Sum2_p;
/* Product: '<S257>/Product5' incorporates:
  * Import: '<Root>/xSunECI' */
Vehicle_B.Product5_d = Vehicle_B.Gain1_j + Vehicle_U.xSunECI[1];
/* Product: '<S257>/Product6' */
Vehicle_B.Product6_b = Vehicle_B.Product1_fv + Vehicle_B.Product1_fv;
/* Product: '<S257>/Product7' */
Vehicle_B.Product7_h = Vehicle_B.Product2_oi + Vehicle_B.Product2_oi;
/* Sum: '<S257>/Sum3' incorporates:
  * Constant: '<S257>/Constant' */
Vehicle_B.Sum3_e = (0.5 - Vehicle_B.Product6_b) - Vehicle.B.Product7_h;
/* Gain: '<S257>/Gain2' */
Vehicle_B.Gain2_b = 2.0 * Vehicle.B.Sum3_e;
/* Product: '<S257>/Product8' incorporates:
  * Import: '<Root>/xSunECI' */
Vehicle_B.Product8_g = Vehicle_B.Gain2_b + Vehicle_U.xSunECI[2];
/* Sum: '<S257>/Sum' */
Vehicle_B.Sum_n = (Vehicle_B.Product4_m + Vehicle_B.Product5_d) +
                    Vehicle_B.Product8_g;
/* SignalConversion generated from '<S130>/Dot Product1' */
Vehicle_B.TmpSignalConversionAtDotProduct[0] = Vehicle.B.Sum_lm;
/* Product: '<S265>/Product' incorporates:
  * Import: '<Root>/q_eeci2b' */
Vehicle_B.Product_jz = Vehicle_U.q_eeci2b[0] + Vehicle_U.q_eeci2b[0];
/* Product: '<S265>/Product1' incorporates:
  * Import: '<Root>/q_eeci2b' */
/* Product: '<S265>/Product2' incorporates:
  * Import: '<Root>/q_eeci2b' */
/* Product: '<S265>/Product3' incorporates:
  * Import: '<Root>/q_eeci2b' */
/* Sum: '<S265>/Sum' */
Vehicle_B.Sum_d5 = ((Vehicle_B.Product_jz + Vehicle_B.Product1_bp) +
                    Vehicle_B.Product2_b) + Vehicle.B.Product3_dh;
/* Sqrt: '<S264>/sqrt' */
Vehicle_B.sqrt_k = std::sqrt(Vehicle_B.Sum_d5);
/* Product: '<S260>/Product2' incorporates:
  * Import: '<Root>/q_eeci2b' */
/* Product: '<S261>/Product6' */
Vehicle_B.Product6_1 = Vehicle_B.Product2_lq + Vehicle_B.Product2_1q;
/* Product: '<S260>/Product3' incorporates:
  * Import: '<Root>/q_eeci2b' */
/* Product : '<S261>/Product7' */
Vehicle_B.Product7_d = Vehicle_B.Product3_h + Vehicle_B.Product3_h;
/* Sum : '<S261>/Sum3' incorporates:
  * Constant : '<S261>/Constant' */
Vehicle_B.Sum3_j = (0.5 - Vehicle_B.Product6_l) - Vehicle_B.Product7_d;
/* Gain : '<S261>/Gain2' */
Vehicle_B.Gain2_f = 2.0 * Vehicle_B.Sum3_j;
/* Product : '<S261>/Product8' */
Vehicle_B.Product8_m = 0.0 * Vehicle_B.Gain2_f;
/* Product : '<S260>/Product1' incorporates:
  * Import : '<Root>/q_eci2b' */
/* Product : '<S261>/Product' */
Vehicle_B.Product1_b1 = Vehicle_B.Product1_b1 * Vehicle_B.Product2_lq;
/* Product : '<S260>/Product' incorporates:
  * Import : '<Root>/q_eci2b' */
Vehicle_B.Product_f = Vehicle_U.q_eci2b[0] / Vehicle_B.sqrt_k;
/* Product : '<S261>/Product1' */
Vehicle_B.Product1_l = Vehicle_B.Product_f * Vehicle_B.Product3_h;
/* Sum : '<S261>/Sum1' */
Vehicle_B.Sum1_d = Vehicle_B.Product1_b1 + Vehicle_B.Product1_l;
/* Gain : '<S261>/Gain' */
Vehicle_B.Gain_j = 2.0 * Vehicle_B.Sum1_d;
/* Product : '<S261>/Product4' */
Vehicle_B.Product4_o = Vehicle_B.Gain_j * 0.0;
/* Product : '<S261>/Product2' */
Vehicle_B.Product2_p = Vehicle_B.Product_f * Vehicle_B.Product2_lq;
/* Product : '<S261>/Product3' */
Vehicle_B.Product3_lx = Vehicle_B.Product1_b1 * Vehicle_B.Product3_h;
/* Sum : '<S261>/Sum2' */
Vehicle_B.Sum2_p3 = Vehicle_B.Product3_lx - Vehicle_B.Product2_p;
/* Gain : '<S261>/Gain1' */
Vehicle_B.Gain1_o = 2.0 * Vehicle_B.Sum2_p3;
/* Product : '<S261>/Product5' */
Vehicle_B.Product5_i = Vehicle_B.Gain1_o;
/* Sum : '<S261>/Sum' */
Vehicle_B.Sum_kk = (Vehicle_B.Product8_m + Vehicle_B.Product4_o) + Vehicle_B.Product5_i;
/* Product : '<S262>/Product' */
Vehicle_B.Product_bg = Vehicle_B.Product1_b1 * Vehicle_B.Product2_lq;
/* Product : '<S262>/Product1' */
Vehicle_B.Product1_l3 = Vehicle_B.Product_f * Vehicle_B.Product3_h;
/* Sum : '<S262>/Sum1' */
Vehicle_B.Sum1_l = Vehicle_B.Product_bg - Vehicle_B.Product1_l3;
/* Gain : '<S262>/Gain' */
Vehicle_B.Gain_i = 2.0 * Vehicle_B.Sum1_l;
/* Product : '<S262>/Product4' */
Vehicle_B.Product4_pt = 0.0 * Vehicle_B.Gain_i;
/* Product : '<S262>/Product6' */
Vehicle_B.Product6_p = Vehicle_B.Product1_b1 * Vehicle_B.Product1_b1;
/* Product : '<S262>/Product7' */
Vehicle_B.Product7_dj = Vehicle_B.Product3_h + Vehicle_B.Product3_h;
/* Sum : '<S262>/Sum3' incorporates:
  * Constant : '<S262>/Constant' */
Vehicle_B.Sum3_o = (0.5 - Vehicle_B.Product6_p) - Vehicle_B.Product7_dj;
/* Gain : '<S262>/Gain2' */
Vehicle_B.Gain2_j = 2.0 * Vehicle_B.Sum3_o;
/* Product : '<S262>/Product8' */
Vehicle_B.Product8_e = Vehicle_B.Gain2_j * 0.0;
/* Product : '<S262>/Product2' */
Vehicle_B.Product2 mf = Vehicle_B.Product_f + Vehicle_B.Product1_b1;
/* Product : '<S262>/Product3' */
Vehicle_B.Product3_k = Vehicle_B.Product2_lq * Vehicle_B.Product3_h;
/* Sum : '<S262>/Sum2' */
Vehicle_B.Sum2_a = Vehicle_B.Product2_mf + Vehicle_B.Product3_k;
/* Gain : '<S262>/Gain1' */
Vehicle_B.Gain1_oi = 2.0 + Vehicle_B.Sum2_a;
Vehicle_B.TmpSignalConversionAtDotProduct[1] = Vehicle_B.Sum_ln;
Vehicle_B.TmpSignalConversionAtDotProduct[0] = Vehicle_B.Sum_kk;
Vehicle_B.Product8_a = Vehicle_B.Gain2_o;
Vehicle_B.Gain2_o = 2.0;

Vehicle_B.Sum3_b = (Vehicle_B.Product4_pt + Vehicle_B.Product8_e) / Vehicle_B.Product5_b;
Vehicle_B.Product5_b = Vehicle_B.Sum1_d1;
Vehicle_B.Sum1_d1 = Vehicle_B.Product_l + Vehicle_B.Product1_id;
Vehicle_B.Product1_id = Vehicle_B.Product_f;
Vehicle_B.Product_f = Vehicle_B.Product5_b;

Vehicle_B.Product3_h = Vehicle_B.Product1_b1;
Vehicle_B.Product1_b1 = Vehicle_B.Product3_h;
Vehicle_B.Product3_h = Vehicle_B.Product5_b;
Vehicle_B.Product5_b = Vehicle_B.Sum2_l;
Vehicle_B.Sum2_l = Vehicle_B.Product3_gx;
Vehicle_B.Product3_gx = Vehicle_B.Product2_f;
Vehicle_B.Product2_f = Vehicle_B.TmpSignalConversionAtDotProduct[1];
Vehicle_B.TmpSignalConversionAtDotProduct[1] = Vehicle_B.Sum_kb;
Vehicle_B.TmpSignalConversionAtDotProd_d[1] = Vehicle_B.Sum_kb;

Vehicle_B.TmpSignalConversionAtDotProd_d[0] = Vehicle_B.Sum_kb;
Vehicle_B.TmpSignalConversionAtDotProd_d[1] = Vehicle_B.Sum_kb;

tmpForInput += tmpForInput_idx_1 * tmpForInput_idx_0;

Vehicle_B.Sum1_d1 = Vehicle_B.Product_l + Vehicle_B.Product1_id;
Vehicle_B.Product1_id = Vehicle_B.Product_f;
Vehicle_B.Product_f = Vehicle_B.Product5_b;
Vehicle_B.Product5_b = Vehicle_B.Sum2_l;
Vehicle_B.Sum2_l = Vehicle_B.Product3_gx;
Vehicle_B.Product3_gx = Vehicle_B.Product2_f;
Vehicle_B.Product2_f = Vehicle_B.TmpSignalConversionAtDotProduct[1];
Vehicle_B.TmpSignalConversionAtDotProduct[1] = Vehicle_B.Sum_kb;
Vehicle_B.TmpSignalConversionAtDotProd_d[1] = Vehicle_B.Sum_kb;

Vehicle_B.TmpSignalConversionAtDotProd_d[0] = Vehicle_B.Sum_kb;
Vehicle_B.TmpSignalConversionAtDotProd_d[1] = Vehicle_B.Sum_kb;

tmpForInput += tmpForInput_idx_1 * tmpForInput_idx_0;

Vehicle_B.Sum1_d1 = Vehicle_B.Product_l + Vehicle_B.Product1_id;
Vehicle_B.Product1_id = Vehicle_B.Product_f;
Vehicle_B.Product_f = Vehicle_B.Product5_b;
Vehicle_B.Product5_b = Vehicle_B.Sum2_l;
Vehicle_B.Sum2_l = Vehicle_B.Product3_gx;
Vehicle_B.Product3_gx = Vehicle_B.Product2_f;
Vehicle_B.Product2_f = Vehicle_B.TmpSignalConversionAtDotProduct[1];
Vehicle_B.TmpSignalConversionAtDotProduct[1] = Vehicle_B.Sum_kb;
Vehicle_B.TmpSignalConversionAtDotProd_d[1] = Vehicle_B.Sum_kb;

Vehicle_B.TmpSignalConversionAtDotProd_d[0] = Vehicle_B.Sum_kb;
Vehicle_B.TmpSignalConversionAtDotProd_d[1] = Vehicle_B.Sum_kb;

tmpForInput += tmpForInput_idx_1 * tmpForInput_idx_0;

Vehicle_B.Sum1_d1 = Vehicle_B.Product_l + Vehicle_B.Product1_id;
Vehicle_B.Product1_id = Vehicle_B.Product_f;
Vehicle_B.Product_f = Vehicle_B.Product5_b;
Vehicle_B.Product5_b = Vehicle_B.Sum2_l;
Vehicle_B.Sum2_l = Vehicle_B.Product3_gx;
Vehicle_B.Product3_gx = Vehicle_B.Product2_f;
Vehicle_B.Product2_f = Vehicle_B.TmpSignalConversionAtDotProduct[1];
Vehicle_B.TmpSignalConversionAtDotProduct[1] = Vehicle_B.Sum_kb;
Vehicle_B.TmpSignalConversionAtDotProd_d[1] = Vehicle_B.Sum_kb;

Vehicle_B.TmpSignalConversionAtDotProd_d[0] = Vehicle_B.Sum_kb;
Vehicle_B.TmpSignalConversionAtDotProd_d[1] = Vehicle_B.Sum_kb;

tmpForInput += tmpForInput_idx_1 * tmpForInput_idx_0;
/* DotProduct : '<S130>/Dot Product2' */
tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProduct_d[2];
tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduct_d[2];
tmpForInput_0 += tmpForInput_idx_1 * tmpForInput_idx_0;

/* DotProduct : '<S130>/Dot Product1' */
Vehicle_B.DotProduct3 = tmpForInput_idx_2;

/* DotProduct : '<S130>/Dot Product1' */
Vehicle_B.DotProduct1 = tmpForInput;

/* Product : '<S130>/Divide' */
Vehicle_B.Divide_1 = Vehicle_B.DotProduct1 + Vehicle_B.DotProduct2;

/* Sqrt : '<S130>/Sqrt3' */
Vehicle_B.Sqrt3 = std::sqrt(Vehicle_B.Divide_1);

/* Product : '<S130>/Divide' */
Vehicle_B.Divide_o = Vehicle_B.DotProduct3 / Vehicle_B.Sqrt3;

/* Bias : '<S130>/Bias' */
Vehicle_B.Bias_k = Vehicle_B.Divide_o - 1.0;

/* Abs : '<S130>/Abs' */
Vehicle_B.Abs = std::abs(Vehicle_B.Bias_k);

/* Bias : '<S130>/Bias1' */
Vehicle_B.Bias1_j = Vehicle_B.Divide_o + 1.0;

/* Abs : '<S130>/Abs1' */
Vehicle_B.Abs1_d = std::abs(Vehicle_B.Bias1_j);

/* If : '<S130>/If' */
if (Vehicle_B.Abs < 1.0E-6) {
    /* Outputs for IAction SubSystem : '<S130>/IAction Subsystem' incorporates:
        * ActionPort : '<S143>/Action Port'
        /}
    // Vehicle_IActionSubsystem(&Vehicle_B.Merge_ip);
    Vehicle_B.Merge_ip = 1.0;
    /* End of Outputs for SubSystem : '<S130>/IAction Subsystem' */
} else if (Vehicle_B.Abs1_d < 1.0E-6) {
    /* Outputs for IAction SubSystem : '<S130>/IAction Subsystem1' incorporates:
        * ActionPort : '<S144>/Action Port'
        /
    // Vehicle_IActionSubsystem1(&Vehicle_B.Merge_ip);
    Vehicle_B.Merge_ip = -1.0;
    /* End of Outputs for SubSystem : '<S130>/IAction Subsystem1' */
} else {
    /* Outputs for IAction SubSystem : '<S130>/IAction Subsystem2' incorporates:
        * ActionPort : '<S145>/Action Port'
        /
    // Vehicle_IActionSubsystem2(&Vehicle_B.Merge_ip);
    Vehicle_B.Merge_ip = 0.0;
    /* End of Outputs for SubSystem : '<S130>/IAction Subsystem2' */
}

/* End of If : '<S130>/If' */
/* RelationalOperator : '<S131>/Compare' incorporates:
    * Constant : '<S131>/Constant'
    */
Vehicle_B.Compare_k = (Vehicle_B.Merge_ip != 0.0);

/* RelationalOperator : '<S132>/Compare' incorporates:
    * Constant : '<S132>/Constant'
    */
Vehicle_B.Compare_a = (Vehicle_B.Merge_ip != 0.0);

/* Logic : '<S124>/OR' */
Vehicle_B.OR = (Vehicle_B.Compare_k || Vehicle_B.Compare_a);

/* DotProduct : '<S136>/Dot Product3' */
tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduct[0];
tmpForInput_idx_1 = 0.0 * tmpForInput_idx_0;

/* DotProduct : '<S136>/Dot Product2' */
tmpForInput_idx_2 = Vehicle_B.TmpSignalConversionAtDotProduct[0];
tmpForInput_idx_1 = 0.0 * tmpForInput_idx_0;

/* DotProduct : '<S136>/Dot Product1' */
tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduct[0];
tmpForInput_0 += tmpForInput_idx_1;

/* DotProduct : '<S136>/Dot Product2' */
tmpForInput_idx_2 = Vehicle_B.TmpSignalConversionAtDotProduct[0];
tmpForInput_0 *= tmpForInput_idx_2;

/* DotProduct : '<S136>/Dot Product1' */
tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduct[0];
tmpForInput_0 += tmpForInput_idx_0;

/* DotProduct : '<S136>/Dot Product2' */
tmpForInput_idx_2 = Vehicle_B.TmpSignalConversionAtDotProduct[0];
tmpForInput_0 *= tmpForInput_idx_2;

/* DotProduct : '<S136>/Dot Product1' */
tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduct[0];
tmpForInput_0 += tmpForInput_idx_0;
Vehicle_B.Sum_k;

if (Vehicle_B.Abs_o < 1.0E-6) {
    /* Outputs for IfAction SubSystem: '<S163>/If Action Subsystem' incorporates:
       * ActionPort: '<S165>/Action Port'
       //Vehicle_IFActionSubsystem(&Vehicle_B.Merge_d);
       Vehicle_B.Merge_d = 1.0;
    } else if (Vehicle_B.Abs1_l < 1.0E-6) {
        /* Outputs for IfAction SubSystem: '<S163>/If Action Subsystem1' incorporates:
           * ActionPort: '<S170>/Action Port'
           //Vehicle_IFActionSubsystem1(&Vehicle_B.Merge_d);
           Vehicle_B.Merge_d = -1.0;
        } else {
            /* Outputs for IfAction SubSystem: '<S163>/If Action Subsystem2' incorporates:
               * ActionPort: '<S171>/Action Port'
               //Vehicle_IFActionSubsystem2(&Vehicle_B.Merge_d);
               Vehicle_B.Merge_d = 0.0;
            } */
    } */

    /* End of If: '<S163>/If' */

    /* RelationalOperator: '<S164>/Compare' incorporates:
     * Constant: '<S164>/Constant'
     */
    Vehicle_B.Compare_m = (Vehicle_B.Merge_d != -1.0);

    /* Switch: '<S136>/is 180deg Rot' */
    if (Vehicle_B.Compare_m) {
        /* Product: '<S168>/j x i_k' */
        Vehicle_B.jxi_k = 0.0 * Vehicle_B.Sum_k;
Vehicle_B.Product8_k = Vehicle_ConstB.Sum_d0[0]
Vehicle_B.Product8_k = Vehicle_ConstB.Sum_hm;

Vehicle_B.Gain2_p = 2.0

Vehicle_B.Sum3_b5 = (0.5 * Vehicle_B.Product3_cc) / Vehicle_B.sqrt_mv;

Vehicle_B.Product6_e = Vehicle_B.Product2_e / Vehicle_B.sqrt_mv;

Vehicle_B.Product2_fh + Vehicle_B.Product3_l0;

Vehicle_B.Sum_n3 = ((Vehicle_B.Product_l5 + Vehicle_B.Product1_lg) + Vehicle_B.Product3_l0) / Vehicle_B.Sum_ff[2];


Vehicle_B.Product2_fh = Vehicle_B.Product2_a / Vehicle_B.sqrt_h;

Vehicle_B.Product2_fh + Vehicle_B.Product3_aj;

Vehicle_B.Sum_c = (Vehicle_B.Product_i2 + Vehicle_B.Product1_g) + Vehicle_B.Product2_g) + Vehicle_B.Product3_aj;

Vehicle_B.sqrt_h = std::sqrt(Vehicle_B.Sum_c);

Vehicle_B.Product_fv = Vehicle_B.Add2 / Vehicle_B.sqrt_h;


Vehicle_B.Sum_n3 = ((Vehicle_B.Product_i2 + Vehicle_B.Product1_g) + Vehicle_B.Product2_fh) + Vehicle_B.Product3_l0;

Vehicle_B.Sum_n3 = (Vehicle_B.Product_l5 + Vehicle_B.Product1_lg) + Vehicle_B.Product2_fh) + Vehicle_B.Product3_l0;

Vehicle_B.Sqrt = std::sqrt(Vehicle_B.Sum_n3);

Vehicle_B.Sqrt_mv = std::sqrt(Vehicle_B.Sum_n3);

Vehicle_B.Sqrt_mv = Vehicle_B.Sqrt / Vehicle_B.Sqrt_mv;

Vehicle_B.Product7_a = Vehicle_B.Product7_a / Vehicle_B.Sqrt_mv;


Vehicle_B.Sum3_b5 = (0.5 - Vehicle_B.Product6_e) - Vehicle_B.Product7_a;

Gain = 'S151'/Gain2'

Vehicle_B.Gain2_p = 2.0 * Vehicle_B.Sum3_b5;

Vehicle_B.Product8_k = Vehicle_ConstB.Sum_d0[0] * Vehicle_B.Gain2_p;
Vehicle_B.Gain_kp = 2.0
Vehicle_B.Sum1_h = Vehicle_B.Product_ew + Vehicle_B.Product1_m;
Vehicle_B.Product1_m = Vehicle_B.Product_m
Vehicle_B.Product_ew = Vehicle_B.Product1_l0
Vehicle_B.Product5_l;
Vehicle_B.Sum_oa = (Vehicle_B.Product4_a + Vehicle_B.Product8_me) +
Vehicle_B.Product5_l;
Vehicle_B.Gain1_ox = 2.0
Vehicle_B.Sum2_la = Vehicle_B.Product2_fhq + Vehicle_B.Product3_n;
Vehicle_B.Gain_b = 2.0
Vehicle_B.Sum1_m;
Vehicle_B.Product1_ew = Vehicle_B.Product_m
Vehicle_B.Product1_l0;
Vehicle_B.Product4_a = Vehicle_B.Product1_n / Vehicle_B.sqrt_mv;
Vehicle_B.Product4_c = Vehicle_B.Gain_i + Vehicle_ConstB.Sum_d0[1];
Vehicle_B.Product2_ek = Vehicle_B.Product_m + Vehicle_B.Product2_ke;
Vehicle_B.Product5_i = Vehicle_B.Product1_l0 + Vehicle_B.Product3_cc;
Vehicle_B.Sum2_la = Vehicle_B.Product2_fhq + Vehicle_B.Product3_n;
Vehicle_B.Gain1_ox = 2.0
Vehicle_B.Sum1_m;
Vehicle_B.Product4_a = Vehicle_B.Product8_me;
Flight Software Source Code

*/
Vehicle_B::sqr_i = std::sqrt(Vehicle_B::sum_gt);
*/
Product::Product2 */
Vehicle_B::Product2_nx = Vehicle_B::Product2_a / Vehicle_B::sqr_i;
*/
Product::Product6 */
Vehicle_B::Product6_a = Vehicle_B::Product2_nx * Vehicle_B::Product2_nx;
*/
Product::Product3 */
Vehicle_B::Product3_ce = Vehicle_B::Product3_pe / Vehicle_B::sqr_i;
*/
Vehicle_B::Gain2_l2 = 2.0;
*/
Vehicle_B::Product7_ha = Vehicle_B::Product3_ce + Vehicle_B::Product3_ce;
*/
Vehicle_B::Sum1_a = Vehicle_B::Product1_a + Vehicle_B::Product1_a;
*/
Vehicle_B::Gain_ay = 2.0;
*/
Vehicle_B::Sum1_k;
*/
Vehicle_B::Product6_a = Vehicle_B::Product6_a / Vehicle_B::sqrt_i;
*/
Vehicle_B::Product1_lr = Vehicle_B::Product1_lr + Vehicle_B::Product1_lr;
*/
Gain: '<S157>/Gain1' */
Vehicle_B::Gain1_c = 2.0 * Vehicle_B::Sum1_a;
*/
Product::Product5 */
Vehicle_B::Product5_jx = Vehicle_B::Product5_jx;
*/
Vehicle_B::Product1_mu = Vehicle_B::Product_pb / Vehicle_B::sqrt_i;
*/
Vehicle_B::Sum3_l = (0.5 - Vehicle_B::Product6_a) - Vehicle_B::Product7_ha;
*/
Gain: '<S157>/Gain2' */
Vehicle_B::Gain2_om = 2.0 * Vehicle_B::Sum3_l;
*/
Vehicle_B::Product2_nz = Vehicle_B::Product2_nz;
*/
Vehicle_B::Product3_pe = Vehicle_B::Product3_pe / Vehicle_B::sqrt_i;
*/
Vehicle_B::Product8_p = Vehicle_B::Sum_pb[0] + Vehicle_B::Gain2_om;
*/
Vehicle_B::Gain2_p = Vehicle_B::Gain2_p;
*/
*/
Gain: '<S157>/Gain3' */
Vehicle_B::Gain3_k = Vehicle_B::Product_pe - Vehicle_B::Product1_mu;
*/
Vehicle_B::Gain3_l = Vehicle_B::Product_pe - Vehicle_B::Product1_mu;
*/
Vehicle_B::Gain3 ay = 2.0 * Vehicle_B::Sum3_l;
*/
Vehicle_B::Product4_1 = Vehicle_B::Product4_1 + Vehicle_B::Product5_jx;
*/
Vehicle_B::Product1_lr = Vehicle_B::Product1_lr + Vehicle_B::Product1_lr;
*/
Gain: '<S157>/Gain4' */
Vehicle_B::Gain4_c = 2.0 * Vehicle_B::Sum1_a;
*/
Product::Product4 */
Vehicle_B::Product4_ok = Vehicle_B::Sum_pb[0] + Vehicle_B::Gain_ay;
*/
Vehicle_B::Gain4 ok = Vehicle_B::Gain4 ok;
*/
Vehicle_B::Product6_ld = Vehicle_B::Product6_ld;
*/
Gain: '<S158>/Gain1' */
Vehicle_B::Gain1_c = 2.0 * Vehicle_B::Sum1_a;
*/
Gain: '<S158>/Gain2' */
Vehicle_B::Gain2_om = 2.0 * Vehicle_B::Sum3_l;
*/
Vehicle_B::Product2_nz = Vehicle_B::Product2_nz;
*/
Vehicle_B::Product3_pe = Vehicle_B::Product3_pe / Vehicle_B::sqrt_i;
*/
Vehicle_B::Gain2_p = Vehicle_B::Gain2_p;
*/
*/
Gain: '<S158>/Gain3' */
Vehicle_B::Gain3_k = Vehicle_B::Product_pe - Vehicle_B::Product1_mu;
*/
Vehicle_B::Gain3 l = Vehicle_B::Product_pe - Vehicle_B::Product1_mu;
*/
Vehicle_B::Gain3 ay = 2.0 * Vehicle_B::Sum3_l;
*/
Gain: '<S158>/Gain4' */
Vehicle_B::Gain4_c = 2.0 * Vehicle_B::Sum1_a;
*/
Vehicle_B::Product4_1 = Vehicle_B::Product4_1 + Vehicle_B::Product5_jx;
*/
Vehicle_B::Product1_lr = Vehicle_B::Product1_lr + Vehicle_B::Product1_lr;
*/
Gain: '<S158>/Gain5' */
Vehicle_B::Gain5_a = 2.0 * Vehicle_B::Sum2_m;
*/
Product::Product5 */
Vehicle_B::Product5_jx = Vehicle_B::Product5_jx;
*/
Vehicle_B::Product1_mu = Vehicle_B::Product_pb / Vehicle_B::sqrt_i;
*/
Vehicle_B::Gain2_om = 2.0;
*/
Vehicle_B::Sum3_l = (0.5 - Vehicle_B::Product6_a) - Vehicle_B::Product7_ha;
*/
Gain: '<S157>/Gain2' */
Vehicle_B::Gain2_om = 2.0 * Vehicle_B::Sum3_l;
*/
Vehicle_B::Product2_nz = Vehicle_B::Product2_nz;
*/
Vehicle_B::Product3_pe = Vehicle_B::Product3_pe / Vehicle_B::sqrt_i;
*/
Vehicle_B::Gain2_p = Vehicle_B::Gain2_p;
*/
*/
Gain: '<S157>/Gain3' */
Vehicle_B::Gain3_k = Vehicle_B::Product_pe - Vehicle_B::Product1_mu;
*/
Vehicle_B::Gain3 l = Vehicle_B::Product_pe - Vehicle_B::Product1_mu;
*/
Vehicle_B::Gain3 ay = 2.0 * Vehicle_B::Sum3_l;
*/
Gain: '<S158>/Gain1' */
Vehicle_B::Gain1_c = 2.0 * Vehicle_B::Sum1_a;
*/
Product::Product4 */
Vehicle_B::Product4_ok = Vehicle_B::Sum_pb[0] + Vehicle_B::Gain_ay;
*/
Vehicle_B::Gain4 ok = Vehicle_B::Gain4 ok;
*/
Vehicle_B::Product6_ld = Vehicle_B::Product6_ld;
*/
Gain: '<S158>/Gain2' */
Vehicle_B::Gain2_om = 2.0 * Vehicle_B::Sum3_l;
*/
Vehicle_B::Product2_nz = Vehicle_B::Product2_nz;
*/
Vehicle_B::Product3_pe = Vehicle_B::Product3_pe / Vehicle_B::sqrt_i;
*/
Vehicle_B::Gain2_p = Vehicle_B::Gain2_p;
*/
*/
Gain: '<S158>/Gain3' */
Vehicle_B::Gain3_k = Vehicle_B::Product_pe - Vehicle_B::Product1_mu;
*/
Vehicle_B::Gain3 l = Vehicle_B::Product_pe - Vehicle_B::Product1_mu;
*/
Vehicle_B::Gain3 ay = 2.0 * Vehicle_B::Sum3_l;
*/
Gain: '<S158>/Gain4' */
Vehicle_B::Gain4_c = 2.0 * Vehicle_B::Sum1_a;
*/ Sum: '<S158>/Sum2' */
Vehicle_B.Sum2_mr = Vehicle_B.Product2_cc + Vehicle_B.Product5_n2;
*/ Gain: '<S158>/Gain1' */
Vehicle_B.Gain1_ni = 2.0 * Vehicle_B.Sum2_mr;
*/ Product: '<S158>/Product5' */
Vehicle_B.Product5_a = Vehicle_B.Gain1_n * Vehicle_B.Sum_pb[2];
*/ Sum: '<S158>/Sum' */
Vehicle_B.Sum_gz = (Vehicle_B.Product4_ok + Vehicle_B.Product8_d) + Vehicle_B.Product5_a;
*/ Product: '<S158>/Product1' */
Vehicle_B.Product1_am = Vehicle_B.Product1_a + Vehicle_B.Product3_ce;
*/ Product: '<S158>/Product4' */
*/ Product: '<S158>/Product2' */
Vehicle_B.Product2_cc = Vehicle_B.Product_pb * Vehicle_B.Product1_a;
*/ Product: '<S158>/Product3' */
Vehicle_B.Product3_cd = Vehicle_B.Product2_nx + Vehicle_B.Product3_ce;
*/ Sum: '<S158>/Sum2' */
Vehicle_B.Sum2_o = Vehicle_B.Product3_cd - Vehicle_B.Product2_cc;
*/ Gain: '<S158>/Gain1' */
Vehicle_B.Gain1_ni = 2.0 * Vehicle_B.Sum2_o;
*/ Product: '<S158>/Product5' */
Vehicle_B.Product5_n2 = Vehicle_B.Gain1_ni + Vehicle_B.Sum_pb[1];
*/ Product: '<S158>/Product6' */
Vehicle_B.Product6_d5 = Vehicle_B.Product1_a * Vehicle_B.Product1_a;
*/ Product: '<S158>/Product7' */
Vehicle_B.Product7_em = Vehicle_B.Product2_nx + Vehicle_B.Product2_nx;
*/ Sum: '<S158>/Sum3' incorporates:
  Constant: '<S158>/Constant' */
Vehicle_B.Sum3_jh = (0.5 - Vehicle_B.Product6_d5) - Vehicle_B.Product7_em;
*/ Gain: '<S158>/Gain2' */
Vehicle_B.Gain2 Ja = 2.0 * Vehicle_B.Sum3_jh;
*/ Product: '<S158>/Product8' */
Vehicle_B.Product8_j = Vehicle_B.Product2_o5 + Vehicle_B.Product5_n2 + Vehicle_B.Product8_j;
*/ SignalConversion generated from: '<S178>/Dot Product2' */
Vehicle_B.TmpSignalConversionAtDotProdu_j[0] = Vehicle_B.B.Sum_g5;
Vehicle_B.TmpSignalConversionAtDotProdu_j[1] = Vehicle_B.Sum_gx;
*/ DotProduct: '<S178>/Dot Product4' */
tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProdu_k[0];
tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_j[0];
tmpForInput_idx_2 = tmpForInput_idx_1 * tmpForInput_idx_0;
*/ DotProduct: '<S178>/Dot Product1' */
tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProdu_k[0];
tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_k[0];
tmpForInput = tmpForInput_idx_1 * tmpForInput_idx_0;
*/ DotProduct: '<S178>/Dot Product2' */
tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProdu_k[1];
tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_k[1];
tmpForInput = tmpForInput_idx_1 * tmpForInput_idx_0;
*/ DotProduct: '<S178>/Dot Product3' */
tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProdu_k[1];
tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_k[1];
tmpForInput = tmpForInput_idx_1 * tmpForInput_idx_0;
*/ DotProduct: '<S178>/Dot Product1' */
tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProdu_k[1];
tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_k[1];
tmpForInput = tmpForInput_idx_1 * tmpForInput_idx_0;
*/ DotProduct: '<S178>/Dot Product2' */
tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProdu_k[2];
tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_k[2];
tmpForInput_idx_2 = tmpForInput_idx_1 * tmpForInput_idx_0;
/* DotProduct : '<S178>/Dot Product1' */
tmpForInput_idx_1 += Vehicle_B.TmpSignalConversionAtDotProdukB[2];
tmpForInput_idx_0 += Vehicle_B.TmpSignalConversionAtDotProdukB[2];
tmpForInput += tmpForInput_idx_1 * tmpForInput_idx_0;
/* DotProduct : '<S178>/Dot Product2' */
tmpForInput_idx_1 += Vehicle_B.TmpSignalConversionAtDotProdukJ[2];
tmpForInput_idx_0 += Vehicle_B.TmpSignalConversionAtDotProdukJ[2];
tmpForInput_0 += tmpForInput_idx_1 * tmpForInput_idx_0;
/* DotProduct : '<S178>/Dot Product3' */
Vehicle_B.DotProduct3_ph = tmpForInput_idx_2;
/* DotProduct : '<S178>/Dot Product1' */
Vehicle_B.DotProduct1_o = tmpForInput;
/* DotProduct : '<S178>/Dot Product2' */
Vehicle_B.DotProduct2_g = tmpForInput_0;
/* Product : '<S178>/Divide1' */
Vehicle_B.Divide1_p = Vehicle_B.DotProduct1_o * Vehicle_B.DotProduct2_g;
/* Sqrt : '<S178>/Sqrt4' */
Vehicle_B.Sqrt3_m = std::sqrt(Vehicle_B.Divide1_p);
/* Product : '<S178>/Divide' */
Vehicle_B.Divide_c = Vehicle_B.DotProduct3_ph / Vehicle_B.Sqrt3_m;
/* Bias : '<S178>/Bias1' */
Vehicle_B.Bias_h = Vehicle_B.Divide_c + -1.0;
/* Abs : '<S178>/Abs1' */
Vehicle_B.Abs_m = std::abs(Vehicle_B.Bias_h);
/* Bias : '<S178>/Bias1' */
Vehicle_B.Abs1_e = Vehicle_B.Divide_c + 1.0;
/* Abs : '<S178>/Abs1' */
Vehicle_B.Abs1_i = std::abs(Vehicle_B.Bias1_e);
/* If : '<S178>/If' */
if (Vehicle_B.Abs_m < 1.0E-6) {
    /* Outputs for IAction SubSystem : '<S178>/IAction SubSystem' incorporates:
      * ActionPort : '<S184>/Action Port'
    */
    Vehicle_B.IActionSubSystem(&Vehicle_B.Merge_l);
    Vehicle_B.Merge_l = 1.0;
    /* End of Outputs for SubSystem : '<S178>/IAction SubSystem' */
} else if (Vehicle_B.Abs1_i < 1.0E-6) {
    /* Outputs for IAction SubSystem1 : '<S178>/IAction SubSystem1' incorporates:
      * ActionPort : '<S185>/Action Port'
    */
    Vehicle_B.IActionSubSystem1(&Vehicle_B.Merge_l);
    Vehicle_B.Merge_l = -1.0;
    /* End of Outputs for SubSystem : '<S178>/IAction SubSystem1' */
} else {
    /* Outputs for IAction SubSystem : '<S178>/IAction SubSystem2' incorporates:
      * ActionPort : '<S186>/Action Port'
    */
    Vehicle_B.IActionSubSystem2(&Vehicle_B.Merge_l);
    Vehicle_B.Merge_l = 0.0;
    /* End of Outputs for SubSystem : '<S178>/IAction SubSystem2' */
}
/* End of If : '<S178>/If' */
/* Switch : '<S124>/Switch' */
if (Vehicle_B.OR) {
    /* Switch : '<S124>/Switch' */
    Vehicle_B.Switch_c[0] = Vehicle_B.Product_fv;
    Vehicle_B.Switch_c[1] = Vehicle_B.Product1_n;
    Vehicle_B.Switch_c[2] = Vehicle_B.Product2_a;
    Vehicle_B.Switch_c[3] = Vehicle_B.Product3_pe;
} else {
    /* RelationalOperator : '<S179>/Compare' incorporates:
       * Constants : '<S179>/Constant'
    */
    Vehicle_B.Compare_g = (Vehicle_B.Merge_l != -1.0);
    /* Switch : '<S137>/is 180deg Rot' */
    if (Vehicle_B.Compare_g) {
        /* Product : '<S183>/j x l' */
        Vehicle_B.jxi_no = Vehicle_B.Sum_g5 + Vehicle_B.Sum_g5;
        /* Product : '<S183>/j x k' */
        Vehicle_B.ikx_b = Vehicle_B.Sum_gz + Vehicle_B.Sum_pw;
        /* Product : '<S183>/k x j' */
        Vehicle_B.kjx_n = Vehicle_B.Sum_gz + Vehicle_B.Sum_gz;
        /* Product : '<S182>/j x j' */
        Vehicle_B.jxj_l = Vehicle_B.Sum_gz + Vehicle_B.Sum_gz;
        /* Product : '<S182>/k x i' */
        Vehicle_B.kxi_o = Vehicle_B.Sum_gz + Vehicle_B.Sum_gz;
        /* Product : '<S182>/j x k' */
        Vehicle_B.jxk_d = Vehicle_B.Sum_gz + Vehicle_B.Sum_pw;
        /* Sum : '<S177>/Sum' */
        Vehicle_B.Sum_b[0] = Vehicle_B.jxk_d + Vehicle_B.kjx_n;
        Vehicle_B.Sum_b[1] = Vehicle_B.kxi_o + Vehicle_B.ikx_b;
*/ Switch: '<S137>/is 180deg Rot' */
Vehicle_B.is180degRot[0] = Vehicle_B.Sum_b[0];
Vehicle_B.is180degRot[1] = Vehicle_B.Sum_b[1];
Vehicle_B.is180degRot[2] = Vehicle_B.Sum_b[2];
} else {
    */ RelationalOperator: '<S180>/>z' */
    Vehicle_B.xz = (Vehicle_B.Sum_oa > Vehicle_B.Sum_nx);
*/ Switch: '<S180>/Switch3' */
if (Vehicle_B.xz) {
    */ Gain: '<S180>/Gain' */
    Vehicle_B.Gain_o2 = ~Vehicle_B.Sum_oa;
*/ Switch: '<S180>/Switch3' incorporates:
    */ Constant: '<S180>/Constant' */
    Vehicle_B.Switch3[0] = Vehicle_B.Gain_o2;
    Vehicle_B.Switch3[1] = Vehicle_B.Sum_oa;
    Vehicle_B.Switch3[2] = 0.0;
    } else {
    */ Gain: '<S180>/Gain2' */
    Vehicle_B.Gain2_c = ~Vehicle_B.Sum_nx;
*/ Switch: '<S180>/Switch3' incorporates:
    */ Constant: '<S180>/Constant1' */
    Vehicle_B.Switch3[0] = 0.0;
    Vehicle_B.Switch3[1] = Vehicle_B.Gain2_c;
    Vehicle_B.Switch3[2] = Vehicle_B.Sum_oa;
} } /* End of Switch: '<S180>/Switch3' */
*/ Product: '<S189>/j x i' */
Vehicle_B.jxi_m = Vehicle_B.Sum_oa + Vehicle_B.Switch3[0];
*/ Product: '<S189>/i x k' */
Vehicle_B.kxi_bj = Vehicle_B.Sum_oa + Vehicle_B.Switch3[2];
*/ Product: '<S189>/k x j' */
Vehicle_B.kxj_f = Vehicle_B.Sum_nx + Vehicle_B.Switch3[1];
*/ Product: '<S188>/i x j' */
Vehicle_B.ixj_k = Vehicle_B.Sum_oa + Vehicle_B.Switch3[1];
*/ Product: '<S188>/k x i' */
Vehicle_B.kixji_bj = Vehicle_B.Sum_nx + Vehicle_B.Switch3[0];
*/ Product: '<S188>/j x k' */
Vehicle_B.jxk_l = Vehicle_B.Sum_oa + Vehicle_B.Switch3[2];
*/ Sum: '<S187>/Sum' */
Vehicle_B.Sum_pd[0] = Vehicle_B.jxk_l - Vehicle_B.kxj_f;
*/ Switch: '<S137>/is 180deg Rot' */
Vehicle_B.is180degRot[0] = Vehicle_B.Sum_pd[0];
Vehicle_B.is180degRot[1] = Vehicle_B.Sum_pd[1];
Vehicle_B.is180degRot[2] = Vehicle_B.Sum_pd[2];
} { */ End of Switch: '<S137>/is 180deg Rot' */
*/ Product: '<S191>/Product3' */
Vehicle_B.Product3_dz = Vehicle_B.is180degRot[2] * Vehicle_B.is180degRot[1];
*/ Product: '<S191>/Product2' */
Vehicle_B.Product2_eq = Vehicle_B.is180degRot[1] * Vehicle_B.is180degRot[1];
*/ Product: '<S191>/Product1' */
Vehicle_B.Product1_aa = Vehicle_B.is180degRot[0] * Vehicle_B.is180degRot[0];
*/ DotProduct: '<S137>/Dot Product2' */
tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProduk[0];
tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduk[0];
tmpForInput_idx_2 = tmpForInput_idx_1 * tmpForInput_idx_0;
*/ DotProduct: '<S137>/Dot Product1' */
tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProduk[1] * tmpForInput_idx_1;
tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduk[0] * tmpForInput_idx_1;
tmpForInput = tmpForInput_idx_1 * tmpForInput_idx_0;
*/ DotProduct: '<S137>/Dot Product3' */
tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProduk[0] * tmpForInput_idx_1;
tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduk[1] * tmpForInput_idx_1;
tmpForInput = tmpForInput_idx_1 * tmpForInput_idx_0;
*/ DotProduct: '<S137>/Dot Product2' */
tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProduk[0] * tmpForInput_idx_1;
tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduk[1] * tmpForInput_idx_1;
tmpForInput = tmpForInput_idx_1 * tmpForInput_idx_0;
*/ DotProduct: '<S137>/Dot Product3' */
tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProduk[1] * tmpForInput_idx_1;
tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduk[0] * tmpForInput_idx_1;
tmpForInput = tmpForInput_idx_1 * tmpForInput_idx_0;
*/ DotProduct: '<S137>/Dot Product3' */
tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProduk[0] * tmpForInput_idx_1;
tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduk[1] * tmpForInput_idx_1;
Flight Software Source Code
Vehicle_B.Product2_i) + Vehicle_B.Product3_h5;

/* Product : '<S149>/Product3' */
Vehicle_B.Product3_c1 = Vehicle_B.Product3_pe * Vehicle_B.Product3_ey;
/* Product : '<S149>/Product2' */
Vehicle_B.Product2_no = Vehicle_B.Product2_a * Vehicle_B.Product1_ge;
/* Product : '<S149>/Product1' */
Vehicle_B.Product1_d1 = Vehicle_B.Product1_n * Vehicle_B.Product2_ep;
/* Product : '<S149>/Product' */
Vehicle_B.Product_gr = Vehicle_B.Product_fv * Vehicle_B.Product3_izz;
/* Sum : '<S149>/Sum' */
Vehicle_B.Sum_nl = (Vehicle_B.Product_gr + Vehicle_B.Product1_d1) -
Vehicle_B.Product2_no) + Vehicle_B.Product3_c1;
/* Switch : '<S124>/Switch' */
Vehicle_B.Switch_c[0] = Vehicle_B.Sum_ki;
Vehicle_B.Switch_c[1] = Vehicle_B.Sum_qk;
Vehicle_B.Switch_c[2] = Vehicle_B.Sum_kc;
Vehicle_B.Switch_c[3] = Vehicle_B.Sum_nl;
*/
*/ End of Switch : '<S124>/Switch' */
/* Product : '<S273>/Product' */
Vehicle_B.Product_h0 = Vehicle_B.Switch_c[0] * Vehicle_B.Switch_c[0];
/* Product : '<S273>/Product1' */
Vehicle_B.Product1_fm = Vehicle_B.Switch_c[1] * Vehicle_B.Switch_c[1];
/* Product : '<S273>/Product2' */
Vehicle_B.Product2_c2 = Vehicle_B.Switch_c[2] * Vehicle_B.Switch_c[2];
/* Product : '<S273>/Product3' */
Vehicle_B.Product3_pq = Vehicle_B.Switch_c[3] * Vehicle_B.Switch_c[3];
/* Sum : '<S273>/Sum' */
Vehicle_B.Sum_f = ((Vehicle_B.Product_h0 + Vehicle_B.Product1_fm) +
Vehicle_B.Product2_c2) + Vehicle_B.Product3_pq;
/* Sqt : '<S272>/sqrt' */
Vehicle_B.sqrt_mo = std::sqrt(Vehicle_B.Sum_f);
/* Product : '<S267>/Product' */
Vehicle_B.Product jc = Vehicle_B.Switch_c[0] / Vehicle_B.sqrt_mo;
/* Product : '<S267>/Product1' */
Vehicle_B.Product1 o = Vehicle_B.Switch_c[1] / Vehicle_B.sqrt_mo;
/* Product : '<S267>/Product2' */
/* Product : '<S267>/Product3' */
/* Fcn : '<S128>/fcn1' */
Vehicle_B.fcn1_i = (Vehicle_B.Product2 j * Vehicle_B.Product3 lz -
Vehicle_B.Product jc) / Vehicle_B.Product1 o) = -2.0;
/* Fcn : '<S128>/fcn2' */
Vehicle_B.fcn2_f = ((Vehicle_B.Product jc * Vehicle_B.Product jc -
Vehicle_B.Product1 o) * Vehicle_B.Product1 o -
Vehicle_B.Product2 j * Vehicle_B.Product2 j) +
Vehicle_B.Product3 lz * Vehicle_B.Product3 lz;
/* Trigonometry : '<S266>/Trigonometric Function1' */
Vehicle_B.Vector Concatenate_a[0] = rt_atan2d_snf(Vehicle_B.fcn1_i,
Vehicle_B.fcn2_f);
/* Fcn : '<S128>/fcn3' */
Vehicle_B.fcn3_o = (Vehicle_B.Product1 o + Vehicle_B.Product3 lz +
Vehicle_B.Product jc) * Vehicle_B.Product2 j) * 2.0;
/* If : '<S268>/If' */
if (Vehicle_B.fcn3_o > 1.0) {
/* Outputs for IFAction SubSystem : '<S268>/IF Action Subsystem2' incorporates:
 * ActionPort : '<S271>/Action Port'
 */
/* Merge : '<S268>/Merge2' incorporates:
 * Constant : '<S269>/Constant'
 */
Vehicle_B.Merge_k = 1.0;
} else if (Vehicle_B.fcn3_o < -1.00) {
/* Outputs for IFAction SubSystem : '<S268>/IF Action Subsystem1' incorporates:
 * ActionPort : '<S270>/Action Port'
 */
/* Merge : '<S268>/Merge1' incorporates:
 * Constant : '<S270>/Constant'
 */
Vehicle_B.Merge_k = 1.0;
} else {
/* Outputs for IFAction SubSystem : '<S268>/IF Action Subsystem2' incorporates:
 * ActionPort : '<S271>/Action Port'
 */
Vehicle_ifActionSubSystem2_f(Vehicle_B.fcn3_o, &Vehicle_B.Merge_k);
Vehicle_B.Merge_k = Vehicle_B.fcn3_o;
/* End of Outputs for SubSystem : '<S268>/IF Action Subsystem2' */
Vehicle_B . ProportionalGain_h [1] = 1.00000000000000E 4
Vehicle_B . IntegralGain_p [1] = 2.0E 4
Vehicle_B . UD_l [1] = Vehicle_DW .UD_DSTATE_d[1];
Vehicle_DW . Integrator_DSTATE_f [0] += Vehicle_B . IntegralGain_p [0];
Vehicle_DW .UD_DSTATE_d[0] = Vehicle_B . Tsamp_f [0];
Vehicle_B . Integrator_k [0] ) + Vehicle_B . Diff_e [0];
Vehicle_B . Merge_k [0] ;
Vehicle_B . VectorConcatenate_a [0] = std :: asin(tmpForInput_idx_0);
}
/* End of If: '<S266>/If' */
/* Trigonometry: '<S266>/trigFcn' */
Vehicle_B . VectorConcatenate_a [1] = rt_atan2d_snf ( Vehicle_B . fcn4_n , Vehicle_B . fcn5_g);
/* Gain: '<S229>/Derivative Gain' */
Vehicle_B . DerivativeGain_k [0] = 0.02 * Vehicle_B . VectorConcatenate_a [0];
/* SampleTimeMath: '<S232>/Tsamp' *
About '<S232>/Tsamp':
+ y = u * K where K = 1 / ( w * Ts )
+ Multiplication by K = weightedTsampQuantized is being
+ done implicitly by changing the scaling of the input signal.
+ No work needs to be done here. Downstream blocks may need
+ to do work to handle the scaling of the output; this happens
+ automatically.
/*
Vehicle_B . Tsamp_f [0] = Vehicle_B . DerivativeGain_k [0];
/* Delay: '<S230>/AD' */
Vehicle_B . UD_l [0] = Vehicle_DW .UD_DSTATE_d[0];
/* Sum: '<S230>/Diff' */
Vehicle_B . Diff_e [0] = Vehicle_B . Tsamp_f [0] - Vehicle_B . UD_l [0];
/* Gain: '<S234>/Integral Gain' */
Vehicle_B . IntegralGain_p [0] = 2.0E-10 * Vehicle_B . VectorConcatenate_a [0];
/* DiscreteIntegrator: '<S237>/Integrator' */
Vehicle_B . Integrator_k [0] = Vehicle_DW . Integrator_DSTATE_f [0];
/* Gain: '<S242>/Proportional Gain' */
Vehicle_B . proportionalGain_h [0] = 1.0E00000000000000E-4 * Vehicle_B . VectorConcatenate_a [0];
/* Merge: '<S1>/Merge' incorporates:
+ Sum: '<S246>/Sum'
*/
Vehicle_B . Merge [0] = (Vehicle_B . ProportionalGain_h [0] + Vehicle_B . Integrator_k [0]) + Vehicle_B . Diff_e [0];
/* Update for Delay: '<S230>/AD' */
Vehicle_DW .UD_DSTATE_d[0] = Vehicle_B . Tsamp_f [0];
/* Update for DiscreteIntegrator: '<S237>/Integrator' */
Vehicle_DW . Integrator_DSTATE_f [0] = Vehicle_B . IntegralGain_p [0];
/* Gain: '<S229>/Derivative Gain' */
Vehicle_B . DerivativeGain_k [1] = 0.02 * Vehicle_B . VectorConcatenate_a [1];
/* SampleTimeMath: '<S232>/Tsamp' *
+ y = u * K where K = 1 / ( w * Ts )
+ Multiplication by K = weightedTsampQuantized is being
+ done implicitly by changing the scaling of the input signal.
+ No work needs to be done here. Downstream blocks may need
+ to do work to handle the scaling of the output; this happens
+ automatically.
/*
/* Delay: '<S230>/AD' */
Vehicle_B . UD_l [1] = Vehicle_DW .UD_DSTATE_d[1];
/* Sum: '<S230>/Diff' */
/* Gain: '<S234>/Integral Gain' */
Vehicle_B . IntegratorGain_p [1] = 2.0E-10 * Vehicle_B . VectorConcatenate_a [1];
/* DiscreteIntegrator: '<S237>/Integrator' */
/* Gain: '<S242>/Proportional Gain' */
Vehicle_B . ProportionalGain_h [1] = 1.00000000000000E-4 *
Vehicle.h

/* Vehicle.h */
/*
   * Academic License — for use in teaching, academic research, and meeting
course requirements at degree granting institutions only. Not for
government, commercial, or other organizational use.
*/

/* Code generation for model 'Vehicle'.

* Model version : 2.125
* Simulink Coder version : R2020b (Jul 2020)
* C++ source code generated on : Mon Nov 2 12:16:23 2020
* Target selection : grt.tlc
* Note: GRT includes extra infrastructure and instrumentation for prototyping
* Embedded hardware selection : 32-bit Generic
* Code generation objective : Debugging
* Validation result: Not run
*/

#define RTW_HEADER_Vehicle_h
#define RTW_HEADER_Vehicle_h
#include <cmath>
#include <cstring>
#include <stdlib.h>
#include "TripleData.h"
#include "SingleData.h"

typedef void * pointer_T;
/* My types */
typedef double real_T;
typedef unsigned int uint32_T;
typedef int int32_T;
typedef unsigned char boolean_T;
typedef char char_T;
typedef float real32_T;
typedef unsigned int uint16_T;
typedef unsigned char uint8_T;

#define RAD_HARD

typedef struct HardenedData : TripleData<real_T> 
{ Merge[3]; /**< <S11>/Merge */ 
jxk; /**< <S20>/j x k */ 
 kxi; /**< <S20>/k x i */ 
 ijk; /**< <S20>/i x j */ 
kxj; /**< <S21>/k x j */ 
ixk; /**< <S21>/i x k */ 
 jxi; /**< <S21>/j x i */ 
 kxj_o; /**< <S18>/k x j */ 
 kxi_e; /**< <S18>/k x i */ 
 ijk_g; /**< <S18>/i x j */ 
jx_i; /**< <S19>/j x i */ 
 jxi_a; /**< <S19>/j x i */ 
 Sum_[1][3]; /**< <S12>/Sum */ 
 DotProduct_; /**< <S15>/Dot Product */ 
 SumofElements_; /**< <S15>/Sum of Elements */ 
 SumofElements_c_; /**< <S16>/Sum of Elements */ 
 SumofElements_l_; /**< <S17>/Sum of Elements */ 
 Sum_; /**< <S11>/Sum */ 
 Divide_; /**< <S9>/Divide */ 
 UnaryMinus_; /**< <S10>/Unary Minus */ 
 Product1_b; /**< <S9>/Product */ 
 Product1_d; /**< <S9>/Product */ 
 Product1_l; /**< <S11>/Product */ 
 Product1_p; /**< <S9>/Product */ 
 Product2_b; /**< <S12>/Product */ 
 Product2_l; /**< <S12>/Product */ 
 Product2_p; /**< <S12>/Product */ 
 Product3_; /**< <S11>/Product */ 
 Product3_d; /**< <S12>/Product */ 
 Product3_l; /**< <S11>/Product */ 
 Product3_p; /**< <S12>/Product */ 
 Product3_s; /**< <S12>/Product */ 
 Product3_t; /**< <S12>/Product */ 
 Product3_v; /**< <S12>/Product */ 
 Product3_w; /**< <S12>/Product */ 
 Product3_y; /**< <S12>/Product */ 
 Product3_z; /**< <S12>/Product */ 
 Product4_; /**< <S9>/Product */ 
 Product4_d; /**< <S10>/Product */ 
 Product4_l; /**< <S11>/Product */ 
 Product4_p; /**< <S11>/Product */ 
 Product4_s; /**< <S11>/Product */ 
 fcn1_; /**< <S11>/fcn1 */ 
 fcn2_; /**< <S11>/fcn2 */ 
 fcn3_; /**< <S11>/fcn3 */ 
 Merge_H; /**< <S18>/Merge */ 
 Merge_f; /**< <S18>/Merge */ 
 Merge_m; /**< <S18>/Merge */ 
 Merge_n; /**< <S18>/Merge */ 
 Merge_p; /**< <S18>/Merge */ 
 Merge_v; /**< <S18>/Merge */ 
 Merge_x; /**< <S18>/Merge */ 
 Merge_t; /**< <S18>/Merge */ 
 VectorConcatenate_[3]; /**< <S16>/Vector Concatenate */ 
 DerivativeGain_[3]; /**< <S13>/Derivative Gain */ 
 Tsamp_[3]; /**< <S13>/Tsamp */ 
 UD_[3]; /**< <S8>/UD */ 
 Diff_[3]; /**< <S8>/Diff */ 
 IntegralGain_[3]; /**< <S9>/Integral Gain */ 
 Integrator_[3]; /**< <S9>/Integrator */ 
 ProportionalGain_[3]; /**< <S9>/Proportional Gain */ 
 Product_a_; /**< <S9>/Product */ 
 Product_b_; /**< <S9>/Product */ 
 Product_c_; /**< <S9>/Product */ 
 Product_d_; /**< <S9>/Product */ 
 Product_e_; /**< <S9>/Product */ 
 Product5_; /**< <S9>/Product */
HardenedData<TripleData<real_T>> Product6_l;
HardenedData<TripleData<real_T>> Product7_d;
HardenedData<TripleData<real_T>> Sum_j;
HardenedData<TripleData<real_T>> Gain2_f;
HardenedData<TripleData<real_T>> Product8_m;
HardenedData<TripleData<real_T>> Product1_b1;
HardenedData<TripleData<real_T>> Product_f;
HardenedData<TripleData<real_T>> Product1_l;
HardenedData<TripleData<real_T>> Sum1_d;
HardenedData<TripleData<real_T>> Gain_j;
HardenedData<TripleData<real_T>> Product3_l0;
HardenedData<TripleData<real_T>> Product2_fh;
HardenedData<TripleData<real_T>> Product_fv;
HardenedData<TripleData<real_T>> sqrt_h;
HardenedData<TripleData<real_T>> Product3_aj;
HardenedData<TripleData<real_T>> Product1_g;
HardenedData<TripleData<real_T>> is180degRot[3];
HardenedData<TripleData<real_T>> Abs1_l;
HardenedData<TripleData<real_T>> Bias1_m;
HardenedData<TripleData<real_T>> Abs_o;
HardenedData<TripleData<real_T>> Divide_k;
HardenedData<TripleData<real_T>> DotProduct2_f;
HardenedData<TripleData<real_T>> Merge_ip;
HardenedData<TripleData<real_T>> Add2;
HardenedData<TripleData<real_T>> Product12;
HardenedData<TripleData<real_T>> DotProduct3_p;
HardenedData<TripleData<real_T>> Sqrt3;
HardenedData<TripleData<real_T>> Sqrt3_j;
HardenedData<TripleData<real_T>> Add3;
HardenedData<TripleData<real_T>> Merge_d;
HardenedData<TripleData<real_T>> Abs1_l;
HardenedData<TripleData<real_T>> Abs1_d;
HardenedData<TripleData<real_T>> Abs;
HardenedData<TripleData<real_T>> Bias1_j;
HardenedData<TripleData<real_T>> Bias1_s;
HardenedData<TripleData<real_T>> Bias;
HardenedData<TripleData<real_T>> Product6_l;
HardenedData<TripleData<real_T>> Product1_l;
HardenedData<TripleData<real_T>> Product6_j;
HardenedData<TripleData<real_T>> Product6_g;
HardenedData<TripleData<real_T>> Product6_f;
HardenedData<TripleData<real_T>> Product6_e;
HardenedData<TripleData<real_T>> Product6_d;
HardenedData<TripleData<real_T>> Product6_c;
HardenedData<TripleData<real_T>> Product6_b;
HardenedData<TripleData<real_T>> Product6_a;
HardenedData<TripleData<real_T>> Sum2_kb;
HardenedData<TripleData<real_T>> Product8_a;
HardenedData<TripleData<real_T>> Sum_kb;
HardenedData<TripleData<real_T>> Add2;
HardenedData<TripleData<real_T>> Product12;
HardenedData<TripleData<real_T>> DotProduct3_p;
HardenedData::SingleData<real_T> Product3_dh;
HardenedData::SingleData<real_T> TmpSignalConversionAtDotProduct[3];
HardenedData::SingleData<real_T> Sum_n;
HardenedData::SingleData<real_T> Product8_g;

HardenedData::SingleData<real_T> Product5_d;

HardenedData::SingleData<real_T> Gain_k;
HardenedData::SingleData<real_T> Sum1_bu;
HardenedData::SingleData<real_T> Product_c3;

HardenedData::SingleData<real_T> Product3_c5;
HardenedData::SingleData<real_T> Product2_l;
HardenedData::SingleData<real_T> Sum3_p;
HardenedData::SingleData<real_T> Product4_k;

HardenedData::SingleData<real_T> Gain1_l;
HardenedData::SingleData<real_T> Sum2;
HardenedData::SingleData<real_T> Product3_pv;
HardenedData::SingleData<real_T> Product4_p;

Flight Software Source Code
extern void SEU();

// Block signals (default storage) *
extern B_Vehicle_T Vehicle_B;

// Block states (default storage) *
extern DW_Vehicle_T Vehicle_DW;

// External inputs (root import signals with default storage) *
extern ExtU_Vehicle_T Vehicle_U;

// External outputs (root outports fed by signals with default storage) *
extern ExtY_Vehicle_T Vehicle_Y;

//Constants
extern ConstB_Vehicle_T Vehicle_ConstB;

#if defined __cplusplus
}
#endif

/* Real-time Model object */
#endif

extern "C" {
}
#endif

// extern RT_MODEL_Vehicle_T *const Vehicle_M;

#if defined __cplusplus
}
#endif

/* RTW_HEADER_Vehicle.h_ */

#include "Vehicle.h"

cenum {
    Inputs_Noise = 1,
    Outputs_Noise,
    States_Noise,
    Consts_Noise,
    BlockSignals_Noise
};

/*Local Functions Declaration*/
void Inputs_Corruption(int index);
void Outputs_Corruption(int index);
void States_Corruption(int index);
void Consts_Corruption(int index);
void BlockSignals_Corruption(int index);

/*Single Event Upset function which is called from FlightSoftware*/
void SEU()
{
    int selection;
    int index;
    selection = (rand() % 4) + 1; //1–5

    switch (selection)
    {
    case Inputs_Noise:
        index = (rand() % 20) + 1; //1 – 21
        Inputs_Corruption(index);
        break;
    case Outputs_Noise:
        index = (rand() % 5) + 1; //1–6
        Outputs_Corruption(index);
        break;
    case States_Noise:
        index = (rand() % 14) + 1; //1–15
        States_Corruption(index);
        break;
    case Consts_Noise:
        index = (rand() % 43) + 1; //1–44
        Consts_Corruption(index);
        break;
    case BlockSignals_Noise:
        index = (rand() % 613)+1; //1–614
        BlockSignals_Corruption(index);
        break;
    }
}

/*Inputs Corruption function*/
void Inputs_Corruption(int index)
{
    switch (index)
    {
    /*Inputs corruption*/
    }
164 case 33: Vehicle_ConstB . ixj . a ( ) = ( real_T ) index ; break ;

162 case 31: Vehicle_ConstB . ixk . a ( ) = ( real_T ) index ; break ;

161 case 30: Vehicle_ConstB . ixk_c . a ( ) = ( real_T ) index ; break ;

160 case 29: Vehicle_ConstB . ixj_g . a ( ) = ( real_T ) index ; break ;

159 case 28: Vehicle_ConstB . ixj_a . a ( ) = ( real_T ) index ; break ;

158 case 27: Vehicle_ConstB . i xj . a ( ) = ( real_T ) index ; break ;

156 case 25: Vehicle_ConstB . Switch3 [1] . a ( ) = ( real_T ) index ; break ;

155 case 24: Vehicle_ConstB . Switch3 [0] . a ( ) = ( real_T ) index ; break ;

154 case 23: Vehicle_ConstB . Sum_d0 [2] . a ( ) = ( real_T ) index ; break ;

153 case 22: Vehicle_ConstB . Sum_d0 [1] . a ( ) = ( real_T ) index ; break ;

152 case 21: Vehicle_ConstB . Sum_d0 [0] . a ( ) = ( real_T ) index ; break ;


150 case 19: Vehicle_ConstB . Sum_d [1] . a ( ) = ( real_T ) index ; break ;

149 case 18: Vehicle_ConstB . Sum_d [0] . a ( ) = ( real_T ) index ; break ;

148 case 17: Vehicle_ConstB . Sum [2] . a ( ) = ( real_T ) index ; break ;

147 case 16: Vehicle_ConstB . Sum [1] . a ( ) = ( real_T ) index ; break ;

146 case 15: Vehicle_ConstB . Sum [0] . a ( ) = ( real_T ) index ; break ;

145 case 14: Vehicle_ConstB . Sqrt3 . a ( ) = ( real_T ) index ; break ;

144 case 13: Vehicle_ConstB . Gain2 . a ( ) = ( real_T ) index ; break ;

143 case 12: Vehicle_ConstB . Gain . a ( ) = ( real_T ) index ; break ;

142 case 11: Vehicle_ConstB . DotProduct3 . a ( ) = ( real_T ) index ; break ;

141 case 10: Vehicle_ConstB . DotProduct2 . a ( ) = ( real_T ) index ; break ;

140 case 9 : Vehicle_ConstB . DotProduct1_g . a ( ) = ( real_T ) index ; break ;

139 case 8 : Vehicle_ConstB . DotProduct1_c . a ( ) = ( real_T ) index ; break ;

138 case 7 : Vehicle_ConstB . DotProduct1 . a ( ) = ( real_T ) index ; break ;

137 case 6 : Vehicle_ConstB . Divide1 . a ( ) = ( real_T ) index ; break ;

136 case 5 : Vehicle_ConstB . Divide . a ( ) = ( real_T ) index ; break ;

135 case 4 : Vehicle_ConstB . Bias1 . a ( ) = ( real_T ) index ; break ;

134 case 3 : Vehicle_ConstB . Bias . a ( ) = ( real_T ) index ; break ;

131 switch ( index )
130 {
129 switch ( index )
128 {
127 void Consts_Corruption ( int index )
126 {
125 void Consts_Corruption ( int index )
124 {
123 }
122 }
121 case 15: Vehicle_DW .UD_DSTATE_d[2] . a ( ) = ( real_T ) index ; break ;
120 case 14: Vehicle_DW .UD_DSTATE_d[1] . a ( ) = ( real_T ) index ; break ;
119 case 13: Vehicle_DW .UD_DSTATE_d[0] . a ( ) = ( real_T ) index ; break ;
118 case 12: Vehicle_DW .UD_DSTATE[2] . a ( ) = ( real_T ) index ; break ;
117 case 11: Vehicle_DW .UD_DSTATE[1] . a ( ) = ( real_T ) index ; break ;
116 case 10: Vehicle_DW .UD_DSTATE[0] . a ( ) = ( real_T ) index ; break ;
115 case 9 : Vehicle_DW . Integrator_DSTATE_f[2] . a ( ) = ( real_T ) index ; break ;
114 case 8 : Vehicle_DW . Integrator_DSTATE_f[1] . a ( ) = ( real_T ) index ; break ;
113 case 7 : Vehicle_DW . Integrator_DSTATE_f[0] . a ( ) = ( real_T ) index ; break ;
111 case 5 : Vehicle_DW . Integrator_DSTATE[1] . a ( ) = ( real_T ) index ; break ;
110 case 4 : Vehicle_DW . Integrator_DSTATE[0] . a ( ) = ( real_T ) index ; break ;
107 case 1 : Vehicle_DW . state[0] . a ( ) = ( real_T ) index ; break ;
106 {
105 switch ( index )
104 {
103 void States_Corruption ( int index )
102 {
101 void States_Corruption ( int index )
100 {
98 }
97 }
94 case 4 : Vehicle_Y . Commands[0] . a ( ) = ( real_T ) index ; break ;
91 case 1 : Vehicle_Y . Bbody[0] . a ( ) = ( real_T ) index ; break ;
90 /
88 switch ( index )
87 {
86 void Outputs_Corruption ( int index )
85 {
84 void Outputs_Corruption ( int index )
83 {
82 }
81 case 21: Vehicle_U . xSunECI[2] . a ( ) = ( real_T ) index ; break ;
80 case 20: Vehicle_U . xSunECI[1] . a ( ) = ( real_T ) index ; break ;
79 case 19: Vehicle_U . xSunECI[0] . a ( ) = ( real_T ) index ; break ;
77 case 17: Vehicle_U .xECEF[1] . a ( ) = ( real_T ) index ; break ;
76 case 16: Vehicle_U .xECEF[0] . a ( ) = ( real_T ) index ; break ;
75 case 15: Vehicle_U .vECEF[2] . a ( ) = ( real_T ) index ; break ;
74 case 14: Vehicle_U .vECEF[1] . a ( ) = ( real_T ) index ; break ;
73 case 13: Vehicle_U .vECEF[0] . a ( ) = ( real_T ) index ; break ;
70 case 10: Vehicle_U . q_eci2b[1] . a ( ) = ( real_T ) index ; break ;
69 case 9 : Vehicle_U . q_eci2b[0] . a ( ) = ( real_T ) index ; break ;
68 case 8 : Vehicle_U . qECEF2b[3] . a ( ) = ( real_T ) index ; break ;
66 case 6 : Vehicle_U . qECEF2b[1] . a ( ) = ( real_T ) index ; break ;
64 case 4 : Vehicle_U . Mode . a ( ) = ( real_T ) index ; break ;
61 case 1 : Vehicle_U .Bmgm[0] . a ( ) = ( real_T ) index ; break ;
//random number
98 }
case 45: Vehicle_B_ConstB_xz[a] = (real_T) index; break;
}

void BlockSignals_Corruption(int index)
{
    switch (index)
    {
    // Inputs corruption
    case 1: Vehicle_B_Abs[a] = (real_T) index; break;
    case 2: Vehicle_B_Abs1[a] = (real_T) index; break;
    case 3: Vehicle_B_Abs1_d[a] = (real_T) index; break;
    case 4: Vehicle_B_Abs1_t[a] = (real_T) index; break;
    case 5: Vehicle_B_Abs1_l[a] = (real_T) index; break;
    case 6: Vehicle_B_Abs2[0][a] = (real_T) index; break;
    case 7: Vehicle_B_Abs2[1][a] = (real_T) index; break;
    case 8: Vehicle_B_Abs2[2][a] = (real_T) index; break;
    case 9: Vehicle_B_Abs2[3][a] = (real_T) index; break;
    case 10: Vehicle_B_Abs2[4][a] = (real_T) index; break;
    case 11: Vehicle_B_Abs2[5][a] = (real_T) index; break;
    case 12: Vehicle_B_Abs2[6][a] = (real_T) index; break;
    case 13: Vehicle_B_Abs2[7][a] = (real_T) index; break;
    case 14: Vehicle_B_Abs2[a] = (real_T) index; break;
    case 15: Vehicle_B_Abs_m[a] = (real_T) index; break;
    case 16: Vehicle_B_Abs_o[a] = (real_T) index; break;
    case 17: Vehicle_B_Add2[a] = (real_T) index; break;
    case 18: Vehicle_B_Add2_k[a] = (real_T) index; break;
    case 19: Vehicle_B_Add2[a] = (real_T) index; break;
    case 20: Vehicle_B_Add_e[a] = (real_T) index; break;
    case 21: Vehicle_B_Add_d[a] = (real_T) index; break;
    case 22: Vehicle_B_Add_c[a] = (real_T) index; break;
    case 23: Vehicle_B_Add_b[a] = (real_T) index; break;
    case 24: Vehicle_B_Add_a[a] = (real_T) index; break;
    case 25: Vehicle_B_Add_f[a] = (real_T) index; break;
    case 26: Vehicle_B_Add_e[a] = (real_T) index; break;
    case 27: Vehicle_B_Add_g[a] = (real_T) index; break;
    case 28: Vehicle_B_Add_gw[a] = (real_T) index; break;
    case 29: Vehicle_B_Add_k[a] = (real_T) index; break;
    case 30: Vehicle_B_Add[a] = (real_T) index; break;
    case 31: Vehicle_B_Add12[a] = (real_T) index; break;
    case 32: Vehicle_B_Add_m[a] = (real_T) index; break;
    case 33: Vehicle_B_Add1[a] = (real_T) index; break;
    case 34: Vehicle_B_Add_ov[a] = (real_T) index; break;
    case 35: Vehicle_B_Bias[a] = (real_T) index; break;
    case 36: Vehicle_B_Bias1[0][a] = (real_T) index; break;
    case 37: Vehicle_B_Bias1[1][a] = (real_T) index; break;
    case 38: Vehicle_B_Bias1[2][a] = (real_T) index; break;
    case 39: Vehicle_B_Bias1[3][a] = (real_T) index; break;
    case 40: Vehicle_B_Bias1[4][a] = (real_T) index; break;
    case 41: Vehicle_B_Bias1[5][a] = (real_T) index; break;
    case 42: Vehicle_B_Bias1[6][a] = (real_T) index; break;
    case 43: Vehicle_B_Bias1[7][a] = (real_T) index; break;
    case 44: Vehicle_B_Bias1[8][a] = (real_T) index; break;
    case 45: Vehicle_B_Bias1_c[a] = (real_T) index; break;
    case 46: Vehicle_B_Bias1_j[a] = (real_T) index; break;
    case 47: Vehicle_B_Bias1_m[a] = (real_T) index; break;
    case 48: Vehicle_B_Bias1[a] = (real_T) index; break;
    case 49: Vehicle_B_Bias_h[a] = (real_T) index; break;
    case 50: Vehicle_B_Bias_k[a] = (real_T) index; break;
    case 51: Vehicle_B_Bias[a] = (real_T) index; break;
    case 52: Vehicle_B_bdot[1][a] = (real_T) index; break;
    case 53: Vehicle_B_bdot[2][a] = (real_T) index; break;
    case 54: Vehicle_B_DerivativeGain[0][a] = (real_T) index; break;
    case 55: Vehicle_B_DerivativeGain[1][a] = (real_T) index; break;
    case 56: Vehicle_B_DerivativeGain[2][a] = (real_T) index; break;
    case 57: Vehicle_B_DerivativeGain_k[0][a] = (real_T) index; break;
    case 58: Vehicle_B_DerivativeGain_k[1][a] = (real_T) index; break;
    case 59: Vehicle_B_DerivativeGain_k[2][a] = (real_T) index; break;
    case 60: Vehicle_B_Diff[0][a] = (real_T) index; break;
    case 61: Vehicle_B_Diff[1][a] = (real_T) index; break;
    case 62: Vehicle_B_Diff[2][a] = (real_T) index; break;
    case 63: Vehicle_B_Diff_e[0][a] = (real_T) index; break;
    case 64: Vehicle_B_Diff_e[1][a] = (real_T) index; break;
    case 65: Vehicle_B_Diff_e[2][a] = (real_T) index; break;
    case 66: Vehicle_B_Diff_e[3][a] = (real_T) index; break;
    case 67: Vehicle_B_Dipole_x[1][a] = (real_T) index; break;
    case 68: Vehicle_B_Dipole_x[2][a] = (real_T) index; break;
    case 69: Vehicle_B_Dipole_y[0][a] = (real_T) index; break;
    case 70: Vehicle_B_Dipole_y[1][a] = (real_T) index; break;
    case 71: Vehicle_B_Dipole_y[2][a] = (real_T) index; break;
    case 72: Vehicle_B_Dipole_z[1][a] = (real_T) index; break;
    case 73: Vehicle_B_Dipole_z[2][a] = (real_T) index; break;
    case 74: Vehicle_B_Divide1[1][a] = (real_T) index; break;
    case 75: Vehicle_B_Divide1[b][a] = (real_T) index; break;
    case 76: Vehicle_B_Divide1_d[a] = (real_T) index; break;
    case 77: Vehicle_B_Divide1_h[a] = (real_T) index; break;
    case 78: Vehicle_B_Divide1_k[a] = (real_T) index; break;
    case 79: Vehicle_B_Divide1_k[a] = (real_T) index; break;
}
case 80: Vehicle_B.Divide1_p_a() = (real_T)index; break;
case 81: Vehicle_B.Divide2_a() = (real_T)index; break;
case 82: Vehicle_B.Divide3_a() = (real_T)index; break;
case 83: Vehicle_B.Divide_c_a() = (real_T)index; break;
case 84: Vehicle_B.Divide_k_a() = (real_T)index; break;
case 85: Vehicle_B.Divide_o_a() = (real_T)index; break;
case 86: Vehicle_B.DotProduct_a() = (real_T)index; break;
case 87: Vehicle_B.DotProduct1_a() = (real_T)index; break;
case 88: Vehicle_B.DotProduct1_b_a() = (real_T)index; break;
case 89: Vehicle_B.DotProduct1_o_a() = (real_T)index; break;
case 90: Vehicle_B.DotProduct2_a() = (real_T)index; break;
case 91: Vehicle_B.DotProduct2_c_a() = (real_T)index; break;
case 92: Vehicle_B.DotProduct2_f_a() = (real_T)index; break;
case 93: Vehicle_B.DotProduct2_e_a() = (real_T)index; break;
case 94: Vehicle_B.DotProduct2_n_a() = (real_T)index; break;
case 95: Vehicle_B.DotProduct3_a() = (real_T)index; break;
case 96: Vehicle_B.DotProduct3_e_a() = (real_T)index; break;
case 97: Vehicle_B.DotProduct3_o_a() = (real_T)index; break;
case 98: Vehicle_B.DotProduct3_p_a() = (real_T)index; break;
case 99: Vehicle_B.DotProduct3_ph_a() = (real_T)index; break;
case 100: Vehicle_B.DotProduct_b_a() = (real_T)index; break;
case 101: Vehicle_B.DotProduct_p_a() = (real_T)index; break;
case 102: Vehicle_B.ECEFtoORF[0].a() = (real_T)index; break;
case 103: Vehicle_B.ECEFtoORF[1].a() = (real_T)index; break;
case 104: Vehicle_B.ECEFtoORF[2].a() = (real_T)index; break;
case 105: Vehicle_B.ECEFtoORF[3].a() = (real_T)index; break;
case 106: Vehicle_B.ECEFtoORF[4].a() = (real_T)index; break;
case 107: Vehicle_B.ECEFtoORF[5].a() = (real_T)index; break;
case 108: Vehicle_B.ECEFtoORF[6].a() = (real_T)index; break;
case 109: Vehicle_B.ECEFtoORF[7].a() = (real_T)index; break;
case 110: Vehicle_B.ECEFtoORF[8].a() = (real_T)index; break;
case 111: Vehicle_B.Gain.a() = (real_T)index; break;
case 112: Vehicle_B.Gain1.a() = (real_T)index; break;
case 113: Vehicle_B.Gain1_c.a() = (real_T)index; break;
case 114: Vehicle_B.Gain1_d.a() = (real_T)index; break;
case 115: Vehicle_B.Gain1_e.a() = (real_T)index; break;
case 116: Vehicle_B.Gain1_f.a() = (real_T)index; break;
case 117: Vehicle_B.Gain1_g.a() = (real_T)index; break;
case 118: Vehicle_B.Gain1_i.a() = (real_T)index; break;
case 119: Vehicle_B.Gain1_l.a() = (real_T)index; break;
case 120: Vehicle_B.Gain1_n.a() = (real_T)index; break;
case 121: Vehicle_B.Gain1_ni.a() = (real_T)index; break;
case 122: Vehicle_B.Gain1_o.a() = (real_T)index; break;
case 123: Vehicle_B.Gain1_ox.a() = (real_T)index; break;
case 124: Vehicle_B.Gain2_a() = (real_T)index; break;
case 125: Vehicle_B.Gain2_b.a() = (real_T)index; break;
case 126: Vehicle_B.Gain2_c.a() = (real_T)index; break;
case 127: Vehicle_B.Gain2_d.a() = (real_T)index; break;
case 128: Vehicle_B.Gain2_e.a() = (real_T)index; break;
case 129: Vehicle_B.Gain2_f.a() = (real_T)index; break;
case 130: Vehicle_B.Gain2_j.a() = (real_T)index; break;
case 131: Vehicle_B.Gain2 JA.a() = (real_T)index; break;
case 132: Vehicle_B.Gain2_ka() = (real_T)index; break;
case 133: Vehicle_B.Gain2_12.a() = (real_T)index; break;
case 134: Vehicle_B.Gain2_n.a() = (real_T)index; break;
case 135: Vehicle_B.Gain2.t.a() = (real_T)index; break;
case 136: Vehicle_B.Gain2_om.a() = (real_T)index; break;
case 137: Vehicle_B.Gain2_p.a() = (real_T)index; break;
case 138: Vehicle_B.Gain2_ta() = (real_T)index; break;
case 139: Vehicle_B.Gain_a.a() = (real_T)index; break;
case 140: Vehicle_B.Gain_g.a() = (real_T)index; break;
case 141: Vehicle_B.Gain_h.a() = (real_T)index; break;
case 142: Vehicle_B.Gain_j.a() = (real_T)index; break;
case 143: Vehicle_B.Gain_i.a() = (real_T)index; break;
case 144: Vehicle_B.Gain_k.a() = (real_T)index; break;
case 145: Vehicle_B.Gain_kp.a() = (real_T)index; break;
case 146: Vehicle_B.Gain_ka() = (real_T)index; break;
case 147: Vehicle_B.Gain_la() = (real_T)index; break;
case 148: Vehicle_B.Gain_o() = (real_T)index; break;
case 149: Vehicle_B.Gain_02_a() = (real_T)index; break;
case 150: Vehicle_B.Gain_p_a() = (real_T)index; break;
case 151: Vehicle_B.IntegralGain[0].a() = (real_T)index; break;
case 152: Vehicle_B.IntegralGain[1].a() = (real_T)index; break;
case 153: Vehicle_B.IntegralGain[2].a() = (real_T)index; break;
case 154: Vehicle_B.IntegralGain[3].a() = (real_T)index; break;
case 155: Vehicle_B.IntegralGain[4].a() = (real_T)index; break;
case 156: Vehicle_B.IntegralGain[5].a() = (real_T)index; break;
case 157: Vehicle_B.Integrator[0].a() = (real_T)index; break;
case 158: Vehicle_B.Integrator[1].a() = (real_T)index; break;
case 159: Vehicle_B.Integrator[2].a() = (real_T)index; break;
case 160: Vehicle_B.Integrator[k][0].a() = (real_T)index; break;
case 161: Vehicle_B.Integrator_k[1].a() = (real_T)index; break;
case 162: Vehicle_B.Integrator_k[2].a() = (real_T)index; break;
case 163: Vehicle_B.is180degRot[0].a() = (real_T)index; break;
case 164: Vehicle_B.is180degRot[1].a() = (real_T)index; break;
case 165: Vehicle_B.is180degRot[2].a() = (real_T)index; break;
case 166: Vehicle_B.is180degRot_l[-1].a() = (real_T)index; break;
case 167: Vehicle_B.is180degRot_l[-1].a() = (real_T)index; break;
case 168: Vehicle_B.is180degRot_l[0].a() = (real_T)index; break;
case 169: Vehicle_B.is180degRot_l[1].a() = (real_T)index; break;
case 170: Vehicle_B.i1xj_f.a() = (real_T)index; break;
case 171: Vehicle_B.i1xj_h.a() = (real_T)index; break;
case 172: Vehicle_B.i1xj_t.a() = (real_T)index; break;
case 173: Vehicle_B.i1xj_l.a() = (real_T)index; break;
case 174: Vehicle_B.i1xj_n.a() = (real_T)index; break;
case 175: Vehicle_B.i1xj_t.a() = (real_T)index; break;
case 176: Vehicle_B.i1xk.a() = (real_T)index; break;
case 177: Vehicle_B.i1xk_a.a() = (real_T)index; break;
case 178: Vehicle_B.i1xk_e.a() = (real_T)index; break;
case 179: Vehicle_B.i1xk_g.a() = (real_T)index; break;
case 180: Vehicle_B.i1xk_k.a() = (real_T)index; break;
case 181: Vehicle_B.ixk_j.a() = (real_T)index; break;

case 182: Vehicle_B.ixk_k.a() = (real_T)index; break;

case 183: Vehicle_B.MathFunction[0].a() = (real_T)index; break;

case 184: Vehicle_B.MathFunction[1].a() = (real_T)index; break;

case 185: Vehicle_B.MathFunction[2].a() = (real_T)index; break;

case 186: Vehicle_B.MathFunction[3].a() = (real_T)index; break;

case 187: Vehicle_B.MathFunction[4].a() = (real_T)index; break;

case 188: Vehicle_B.MathFunction[5].a() = (real_T)index; break;

case 189: Vehicle_B.MathFunction[6].a() = (real_T)index; break;

case 190: Vehicle_B.MathFunction[7].a() = (real_T)index; break;

case 191: Vehicle_B.MathFunction[8].a() = (real_T)index; break;

case 192: Vehicle_B.MathFunction1.a() = (real_T)index; break;

case 193: Vehicle_B.MathFunction2.a() = (real_T)index; break;

case 194: Vehicle_B.MathFunction3.a() = (real_T)index; break;

case 195: Vehicle_B.Merge[0].a() = (real_T)index; break;

case 196: Vehicle_B.Merge[1].a() = (real_T)index; break;

case 197: Vehicle_B.Merge[2].a() = (real_T)index; break;

case 198: Vehicle_B.Merge_d.a() = (real_T)index; break;

case 199: Vehicle_B.Merge_i0.a() = (real_T)index; break;

case 200: Vehicle_B.Merge_i1.a() = (real_T)index; break;

case 201: Vehicle_B.Merge_i2.a() = (real_T)index; break;

case 202: Vehicle_B.Merge_i3.a() = (real_T)index; break;

case 203: Vehicle_B.Merge_i4.a() = (real_T)index; break;

case 204: Vehicle_B.Merge_ip.a() = (real_T)index; break;

case 205: Vehicle_B.Merge_k.a() = (real_T)index; break;

case 206: Vehicle_B.Merge_l.a() = (real_T)index; break;

case 207: Vehicle_B.Merge_m.a() = (real_T)index; break;

case 208: Vehicle_B.ORFtoECEF[0].a() = (real_T)index; break;

case 209: Vehicle_B.ORFtoECEF[1].a() = (real_T)index; break;

case 210: Vehicle_B.ORFtoECEF[2].a() = (real_T)index; break;

case 211: Vehicle_B.ORFtoECEF[3].a() = (real_T)index; break;

case 212: Vehicle_B.ORFtoECEF[4].a() = (real_T)index; break;

case 213: Vehicle_B.ORFtoECEF[5].a() = (real_T)index; break;

case 214: Vehicle_B.ORFtoECEF[6].a() = (real_T)index; break;

case 215: Vehicle_B.ORFtoECEF[7].a() = (real_T)index; break;

case 216: Vehicle_B.ORFtoECEF[8].a() = (real_T)index; break;

case 217: Vehicle_B.Product.a() = (real_T)index; break;

case 218: Vehicle_B.Product1.a() = (real_T)index; break;

case 219: Vehicle_B.Product1_a.a() = (real_T)index; break;

case 220: Vehicle_B.Product1_aa.a() = (real_T)index; break;

case 221: Vehicle_B.Product1_a1.a() = (real_T)index; break;

case 222: Vehicle_B.Product1_b.a() = (real_T)index; break;

case 223: Vehicle_B.Product1_b0.a() = (real_T)index; break;

case 224: Vehicle_B.Product1_b1.a() = (real_T)index; break;

case 225: Vehicle_B.Product1_b2.a() = (real_T)index; break;

case 226: Vehicle_B.Product1_b3.a() = (real_T)index; break;

case 227: Vehicle_B.Product1_c.a() = (real_T)index; break;

case 228: Vehicle_B.Product1_d.a() = (real_T)index; break;

case 229: Vehicle_B.Product1_di.a() = (real_T)index; break;

case 230: Vehicle_B.Product1_dj.a() = (real_T)index; break;

case 231: Vehicle_B.Product1_e.a() = (real_T)index; break;

case 232: Vehicle_B.Product1_aw.a() = (real_T)index; break;

case 233: Vehicle_B.Product1_f.a() = (real_T)index; break;

case 234: Vehicle_B.Product1_fb.a() = (real_T)index; break;

case 235: Vehicle_B.Product1_fm.a() = (real_T)index; break;

case 236: Vehicle_B.Product1_tv.a() = (real_T)index; break;

case 237: Vehicle_B.Product1_g.a() = (real_T)index; break;

case 238: Vehicle_B.Product1_ge.a() = (real_T)index; break;

case 239: Vehicle_B.Product1_h.a() = (real_T)index; break;

case 240: Vehicle_B.Product1_i.a() = (real_T)index; break;

case 241: Vehicle_B.Product1_id.a() = (real_T)index; break;

case 242: Vehicle_B.Product1_ig.a() = (real_T)index; break;

case 243: Vehicle_B.Product1_ii.a() = (real_T)index; break;

case 244: Vehicle_B.Product1_iu.a() = (real_T)index; break;

case 245: Vehicle_B.Product1_j.a() = (real_T)index; break;

case 246: Vehicle_B.Product1_k.a() = (real_T)index; break;

case 247: Vehicle_B.Product1_l.a() = (real_T)index; break;

case 248: Vehicle_B.Product1_l0.a() = (real_T)index; break;

case 249: Vehicle_B.Product1_l3.a() = (real_T)index; break;

case 250: Vehicle_B.Product1_lg.a() = (real_T)index; break;

case 251: Vehicle_B.Product1_le.a() = (real_T)index; break;

case 252: Vehicle_B.Product1_m.a() = (real_T)index; break;

case 253: Vehicle_B.Product1_ma.a() = (real_T)index; break;

case 254: Vehicle_B.Product1_n.a() = (real_T)index; break;

case 255: Vehicle_B.Product1_o.a() = (real_T)index; break;

case 256: Vehicle_B.Product2.a() = (real_T)index; break;

case 257: Vehicle_B.Product2_a.a() = (real_T)index; break;

case 258: Vehicle_B.Product2_ad.a() = (real_T)index; break;

case 259: Vehicle_B.Product2_b.a() = (real_T)index; break;

case 260: Vehicle_B.Product2_be.a() = (real_T)index; break;

case 261: Vehicle_B.Product2_c.a() = (real_T)index; break;

case 262: Vehicle_B.Product2_c2.a() = (real_T)index; break;

case 263: Vehicle_B.Product2_cc.a() = (real_T)index; break;

case 264: Vehicle_B.Product2_d.a() = (real_T)index; break;

case 265: Vehicle_B.Product2_dj.a() = (real_T)index; break;

case 266: Vehicle_B.Product2_e.a() = (real_T)index; break;

case 267: Vehicle_B.Product2_ea.a() = (real_T)index; break;

case 268: Vehicle_B.Product2_ep.a() = (real_T)index; break;

case 269: Vehicle_B.Product2_eq.a() = (real_T)index; break;

case 270: Vehicle_B.Product2_f.a() = (real_T)index; break;

case 271: Vehicle_B.Product2_fh.a() = (real_T)index; break;

case 272: Vehicle_B.Product2_fbq.a() = (real_T)index; break;

case 273: Vehicle_B.Product2_g.a() = (real_T)index; break;

case 274: Vehicle_B.Product2_gl.a() = (real_T)index; break;

case 275: Vehicle_B.Product2_gk.a() = (real_T)index; break;

case 276: Vehicle_B.Product2_i.a() = (real_T)index; break;

case 277: Vehicle_B.Product2_j.a() = (real_T)index; break;

case 278: Vehicle_B.Product2_k.a() = (real_T)index; break;

case 279: Vehicle_B.Product2_ka.a() = (real_T)index; break;

case 280: Vehicle_B.Product2_k2.a() = (real_T)index; break;

case 281: Vehicle_B.Product2_l.a() = (real_T)index; break;

case 282: Vehicle_B.Product2_lq.a() = (real_T)index; break;
case 385: Vehicle_B.Product2_n.a() = (real_T) index; break;
548 case 342: Vehicle_B.Product4_o5.a() = (real_T) index; break;
549 case 343: Vehicle_B.Product4_ok.a() = (real_T) index; break;
550 case 344: Vehicle_B.Product4_p.a() = (real_T) index; break;
551 case 345: Vehicle_B.Product4_pl.a() = (real_T) index; break;
552 case 346: Vehicle_B.Product5_a.f() = (real_T) index; break;
553 case 347: Vehicle_B.Product5_a.a() = (real_T) index; break;
554 case 348: Vehicle_B.Product5_b.a() = (real_T) index; break;
555 case 349: Vehicle_B.Product5_d.a() = (real_T) index; break;
556 case 350: Vehicle_B.Product5_g.a() = (real_T) index; break;
557 case 351: Vehicle_B.Product5_l.a() = (real_T) index; break;
558 case 352: Vehicle_B.Product5_i.j.a() = (real_T) index; break;
559 case 353: Vehicle_B.Product5_j.p.a() = (real_T) index; break;
560 case 354: Vehicle_B.Product5_jx.a() = (real_T) index; break;
561 case 355: Vehicle_B.Product5_k.a() = (real_T) index; break;
562 case 356: Vehicle_B.Product5_n.a() = (real_T) index; break;
563 case 357: Vehicle_B.Product5_n2.a() = (real_T) index; break;
564 case 358: Vehicle_B.Product5_o.a() = (real_T) index; break;
565 case 359: Vehicle_B.Product6_a.f() = (real_T) index; break;
566 case 360: Vehicle_B.Product6_a.a() = (real_T) index; break;
567 case 361: Vehicle_B.Product6_b.a() = (real_T) index; break;
568 case 362: Vehicle_B.Product6_c.a() = (real_T) index; break;
569 case 363: Vehicle_B.Product6_d.a() = (real_T) index; break;
570 case 364: Vehicle_B.Product6_d5.a() = (real_T) index; break;
571 case 365: Vehicle_B.Product6_e.a() = (real_T) index; break;
572 case 366: Vehicle_B.Product6_j.a() = (real_T) index; break;
573 case 367: Vehicle_B.Product6_l.a() = (real_T) index; break;
574 case 368: Vehicle_B.Product6_ld.a() = (real_T) index; break;
575 case 369: Vehicle_B.Product6_o.a() = (real_T) index; break;
576 case 370: Vehicle_B.Product6_p.a() = (real_T) index; break;
577 case 371: Vehicle_B.Product7_a.f() = (real_T) index; break;
578 case 372: Vehicle_B.Product7_a.a() = (real_T) index; break;
579 case 373: Vehicle_B.Product7_a4.a() = (real_T) index; break;
580 case 374: Vehicle_B.Product7_d.a() = (real_T) index; break;
581 case 375: Vehicle_B.Product7_dj.a() = (real_T) index; break;
582 case 376: Vehicle_B.Product7_e.a() = (real_T) index; break;
583 case 377: Vehicle_B.Product7_el.a() = (real_T) index; break;
584 case 378: Vehicle_B.Product7_em.a() = (real_T) index; break;
585 case 379: Vehicle_B.Product7_g.a() = (real_T) index; break;
586 case 380: Vehicle_B.Product7_h.a() = (real_T) index; break;
587 case 381: Vehicle_B.Product7_ha.a() = (real_T) index; break;
588 case 382: Vehicle_B.Product7_k.a() = (real_T) index; break;
589 case 383: Vehicle_B.Product8_a.f() = (real_T) index; break;
590 case 384: Vehicle_B.Product8_a.a() = (real_T) index; break;
591 case 385: Vehicle_B.Product8_c.a() = (real_T) index; break;
592 case 386: Vehicle_B.Product8_d.a() = (real_T) index; break;
case 583: Vehicle_B.TmpSignalConversionAtDotProdu_d[0].a() = (real_T)index; break;

case 584: Vehicle_B.TmpSignalConversionAtDotProdu_d[1].a() = (real_T)index; break;

case 585: Vehicle_B.TmpSignalConversionAtDotProdu_d[2].a() = (real_T)index; break;

case 586: Vehicle_B.TmpSignalConversionAtDotProdu_d[3].a() = (real_T)index; break;

case 587: Vehicle_B.TmpSignalConversionAtDotProdu_d[4].a() = (real_T)index; break;

case 588: Vehicle_B.TmpSignalConversionAtDotProdu_d[5].a() = (real_T)index; break;

case 589: Vehicle_B.TmpSignalConversionAtDotProdu_d[6].a() = (real_T)index; break;

case 5810: Vehicle_B.TmpSignalConversionAtDotProdu_d[7].a() = (real_T)index; break;

case 5811: Vehicle_B.TmpSignalConversionAtDotProdu_d[8].a() = (real_T)index; break;

case 5812: Vehicle_B.TmpSignalConversionAtDotProdu_d[9].a() = (real_T)index; break;

case 5813: Vehicle_B.TmpSignalConversionAtDotProdu_d[10].a() = (real_T)index; break;

case 5814: Vehicle_B.TmpSignalConversionAtDotProdu_d[11].a() = (real_T)index; break;

case 5815: Vehicle_B.TmpSignalConversionAtDotProdu_d[12].a() = (real_T)index; break;

case 5816: Vehicle_B.TmpSignalConversionAtDotProdu_d[13].a() = (real_T)index; break;

case 5817: Vehicle_B.TmpSignalConversionAtDotProdu_d[14].a() = (real_T)index; break;

case 5818: Vehicle_B.TmpSignalConversionAtDotProdu_d[15].a() = (real_T)index; break;

case 5819: Vehicle_B.TmpSignalConversionAtDotProdu_d[16].a() = (real_T)index; break;

case 5820: Vehicle_B.TmpSignalConversionAtDotProdu_d[17].a() = (real_T)index; break;

case 5821: Vehicle_B.TmpSignalConversionAtDotProdu_d[18].a() = (real_T)index; break;

case 5822: Vehicle_B.TmpSignalConversionAtDotProdu_d[19].a() = (real_T)index; break;

case 5823: Vehicle_B.TmpSignalConversionAtDotProdu_d[20].a() = (real_T)index; break;

case 5824: Vehicle_B.TmpSignalConversionAtDotProdu_d[21].a() = (real_T)index; break;

case 5825: Vehicle_B.TmpSignalConversionAtDotProdu_d[22].a() = (real_T)index; break;

case 5826: Vehicle_B.TmpSignalConversionAtDotProdu_d[23].a() = (real_T)index; break;

case 5827: Vehicle_B.TmpSignalConversionAtDotProdu_d[24].a() = (real_T)index; break;

case 5828: Vehicle_B.TmpSignalConversionAtDotProdu_d[25].a() = (real_T)index; break;

case 5829: Vehicle_B.TmpSignalConversionAtDotProdu_d[26].a() = (real_T)index; break;

case 5830: Vehicle_B.TmpSignalConversionAtDotProdu_d[27].a() = (real_T)index; break;

case 5831: Vehicle_B.TmpSignalConversionAtDotProdu_d[28].a() = (real_T)index; break;

case 5832: Vehicle_B.TmpSignalConversionAtDotProdu_d[29].a() = (real_T)index; break;

case 5833: Vehicle_B.TmpSignalConversionAtDotProdu_d[30].a() = (real_T)index; break;

case 5834: Vehicle_B.TmpSignalConversionAtDotProdu_d[31].a() = (real_T)index; break;

case 5835: Vehicle_B.TmpSignalConversionAtDotProdu_d[32].a() = (real_T)index; break;

case 5836: Vehicle_B.TmpSignalConversionAtDotProdu_d[33].a() = (real_T)index; break;

case 5837: Vehicle_B.TmpSignalConversionAtDotProdu_d[34].a() = (real_T)index; break;

case 5838: Vehicle_B.TmpSignalConversionAtDotProdu_d[35].a() = (real_T)index; break;

case 5839: Vehicle_B.TmpSignalConversionAtDotProdu_d[36].a() = (real_T)index; break;

case 5840: Vehicle_B.TmpSignalConversionAtDotProdu_d[37].a() = (real_T)index; break;

case 5841: Vehicle_B.TmpSignalConversionAtDotProdu_d[38].a() = (real_T)index; break;

case 5842: Vehicle_B.TmpSignalConversionAtDotProdu_d[39].a() = (real_T)index; break;

case 5843: Vehicle_B.TmpSignalConversionAtDotProdu_d[40].a() = (real_T)index; break;

case 5844: Vehicle_B.TmpSignalConversionAtDotProdu_d[41].a() = (real_T)index; break;

case 5845: Vehicle_B.TmpSignalConversionAtDotProdu_d[42].a() = (real_T)index; break;

case 5846: Vehicle_B.TmpSignalConversionAtDotProdu_d[43].a() = (real_T)index; break;

case 5847: Vehicle_B.TmpSignalConversionAtDotProdu_d[44].a() = (real_T)index; break;

case 5848: Vehicle_B.TmpSignalConversionAtDotProdu_d[45].a() = (real_T)index; break;

case 5849: Vehicle_B.TmpSignalConversionAtDotProdu_d[46].a() = (real_T)index; break;

case 5850: Vehicle_B.TmpSignalConversionAtDotProdu_d[47].a() = (real_T)index; break;

case 5851: Vehicle_B.TmpSignalConversionAtDotProdu_d[48].a() = (real_T)index; break;

case 5852: Vehicle_B.TmpSignalConversionAtDotProdu_d[49].a() = (real_T)index; break;

case 5853: Vehicle_B.TmpSignalConversionAtDotProdu_d[50].a() = (real_T)index; break;

case 5854: Vehicle_B.TmpSignalConversionAtDotProdu_d[51].a() = (real_T)index; break;

case 5855: Vehicle_B.TmpSignalConversionAtDotProdu_d[52].a() = (real_T)index; break;

case 5856: Vehicle_B.TmpSignalConversionAtDotProdu_d[53].a() = (real_T)index; break;

case 5857: Vehicle_B.TmpSignalConversionAtDotProdu_d[54].a() = (real_T)index; break;

case 5858: Vehicle_B.TmpSignalConversionAtDotProdu_d[55].a() = (real_T)index; break;

case 5859: Vehicle_B.TmpSignalConversionAtDotProdu_d[56].a() = (real_T)index; break;

case 5860: Vehicle_B.TmpSignalConversionAtDotProdu_d[57].a() = (real_T)index; break;

case 5861: Vehicle_B.TmpSignalConversionAtDotProdu_d[58].a() = (real_T)index; break;

case 5862: Vehicle_B.TmpSignalConversionAtDotProdu_d[59].a() = (real_T)index; break;

case 5863: Vehicle_B.TmpSignalConversionAtDotProdu_d[60].a() = (real_T)index; break;

case 5864: Vehicle_B.TmpSignalConversionAtDotProdu_d[61].a() = (real_T)index; break;

case 5865: Vehicle_B.TmpSignalConversionAtDotProdu_d[62].a() = (real_T)index; break;

case 5866: Vehicle_B.TmpSignalConversionAtDotProdu_d[63].a() = (real_T)index; break;

case 5867: Vehicle_B.TmpSignalConversionAtDotProdu_d[64].a() = (real_T)index; break;

case 5868: Vehicle_B.TmpSignalConversionAtDotProdu_d[65].a() = (real_T)index; break;

case 5869: Vehicle_B.TmpSignalConversionAtDotProdu_d[66].a() = (real_T)index; break;

case 5870: Vehicle_B.TmpSignalConversionAtDotProdu_d[67].a() = (real_T)index; break;

case 5871: Vehicle_B.TmpSignalConversionAtDotProdu_d[68].a() = (real_T)index; break;

case 5872: Vehicle_B.TmpSignalConversionAtDotProdu_d[69].a() = (real_T)index; break;

case 5873: Vehicle_B.TmpSignalConversionAtDotProdu_d[70].a() = (real_T)index; break;

case 5874: Vehicle_B.TmpSignalConversionAtDotProdu_d[71].a() = (real_T)index; break;

case 5875: Vehicle_B.TmpSignalConversionAtDotProdu_d[72].a() = (real_T)index; break;

case 5876: Vehicle_B.TmpSignalConversionAtDotProdu_d[73].a() = (real_T)index; break;

case 5877: Vehicle_B.TmpSignalConversionAtDotProdu_d[74].a() = (real_T)index; break;

case 5878: Vehicle_B.TmpSignalConversionAtDotProdu_d[75].a() = (real_T)index; break;

case 5879: Vehicle_B.TmpSignalConversionAtDotProdu_d[76].a() = (real_T)index; break;

case 5880: Vehicle_B.TmpSignalConversionAtDotProdu_d[77].a() = (real_T)index; break;

case 5881: Vehicle_B.TmpSignalConversionAtDotProdu_d[78].a() = (real_T)index; break;

case 5882: Vehicle_B.TmpSignalConversionAtDotProdu_d[79].a() = (real_T)index; break;

case 5883: Vehicle_B.TmpSignalConversionAtDotProdu_d[80].a() = (real_T)index; break;

case 5884: Vehicle_B.TmpSignalConversionAtDotProdu_d[81].a() = (real_T)index; break;

case 5885: Vehicle_B.TmpSignalConversionAtDotProdu_d[82].a() = (real_T)index; break;


case 586: Vehicle_B.TmpSignalConversionAtDotProdu_k[0].a() = (real_T)index; break;

case 587: Vehicle_B.TmpSignalConversionAtDotProdu_k[1].a() = (real_T)index; break;

case 588: Vehicle_B.TmpSignalConversionAtDotProdu_k[2].a() = (real_T)index; break;

case 589: Vehicle_B.Tsamp[0].a() = (real_T)index; break;

case 590: Vehicle_B.Tsamp[1].a() = (real_T)index; break;

case 591: Vehicle_B.Tsamp[2].a() = (real_T)index; break;

case 592: Vehicle_B.Tsamp_f[0].a() = (real_T)index; break;

case 593: Vehicle_B.Tsamp_f[1].a() = (real_T)index; break;

case 594: Vehicle_B.Tsamp_f[2].a() = (real_T)index; break;

case 595: Vehicle_B.Tsamp_f[0].a() = (real_T)index; break;

case 596: Vehicle_B.Tsamp_f[1].a() = (real_T)index; break;

case 597: Vehicle_B.Tsamp_f[2].a() = (real_T)index; break;

case 598: Vehicle_B.Tsamp_f[0].a() = (real_T)index; break;

case 599: Vehicle_B.Tsamp_f[1].a() = (real_T)index; break;

case 600: Vehicle_B.Tsamp_f[2].a() = (real_T)index; break;

case 601: Vehicle_B.UnaryMinus.a() = (real_T)index; break;

case 602: Vehicle_B.UnaryMinus.a() = (real_T)index; break;

case 603: Vehicle_B.UnaryMinus.a() = (real_T)index; break;

case 604: Vehicle_B.VectorConcatenate[0].a() = (real_T)index; break;

case 605: Vehicle_B.VectorConcatenate[1].a() = (real_T)index; break;

case 606: Vehicle_B.VectorConcatenate[2].a() = (real_T)index; break;

case 607: Vehicle_B.VectorConcatenate_a[0].a() = (real_T)index; break;

case 608: Vehicle_B.VectorConcatenate_a[1].a() = (real_T)index; break;

case 609: Vehicle_B.VectorConcatenate_a[2].a() = (real_T)index; break;

case 610: Vehicle_B.zr[0].a() = (real_T)index; break;

case 611: Vehicle_B.zr[1].a() = (real_T)index; break;

case 612: Vehicle_B.zr[2].a() = (real_T)index; break;

case 614: Vehicle_B.norm_b.a() = (real_T)index; break;
Bibliography


