POLITECNICO DI TORINO

Master's Degree in Electronics Engineering / Embedded Systems

Department of Electronics and Telecommunications

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

3.1.1 Mission Description 13
3.1.2 Orbital Parameters 14
3.2 Environment 15
3.2.1 Gravitational Model 15
3.2.2 Sun Position Model 15
3.2.3 Magnetic Field Model 16
3.2.4 Disturbances 16
3.3 Satellite Dynamics 18
3.4 Sensors 22
3.4.1 Magnetometer 22
3.4.2 Gyroscope 23
3.4.3 Sun Sensor 23
3.4.4 GPS 25
3.5 Actuators 26
3.5.1 Magnetorquer Rods 27
3.5.2 Reaction Wheels 28
3.6 Flight Software 30
3.6.1 Detumbling Mode 31
3.6.2 Sun Pointing Mode 32
3.6.3 Nadir Pointing Mode 35
3.6.4 Visualization 35

4 Radiations Hardened Software 38
4.1 Development of Flight Software 38
4.1.1 Data Structures 39
4.1.2 Software Hierarchy 39
4.2 Development of Radiation Hardened Flight Software 40
4.2.1 Radiation Effects on Software 40
4.2.2 Radiation Hardening Library 41
4.2.3 Data Structures 42
4.2.4 Software Hierarchy 42
4.2.5 Radiation Effects Sub-Routine 43
4.2.6 Radiation Tolerance 43
4.2.7 S-functions Implementation 43

5 Simulations and Results 45
5.1 Matlab & Simulink Results 45
5.1.1 Development Platforms & Tools 45

5.1.2 Mission Modes 45
5.2 Radiation Effects 48
5.2.1 Mission Modes Simulation Results 50
5.3 Radiation Hardened Software 52
5.3.1 Mission Modes Simulation Results 52
5.4 Comparative Analysis 55

6 Conclusion 59
6.1 Future Work 60

A Flight Software Source Code 61

Bibliography 111

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 Simuink Block 24
3.9 GPS representation of Simuink Block 26
3.10 Magnetorquer Rods model Simulink representation 28
3.11 Tetrahedron configuration of Reaction Wheels[2] 29
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 Simulation 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 Simulation 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 radiations 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:

$$HardenedData :: TripleData < DataType > Variable; \qquad (1.1)$$

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-hand Cartesian coordinate system[2]. 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.[3]

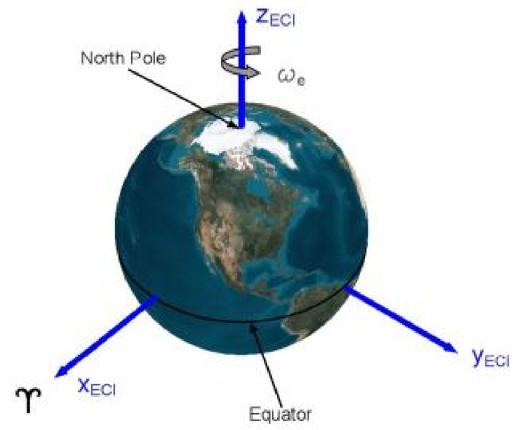

Figure 2.1: ECI Coordinate System[3]

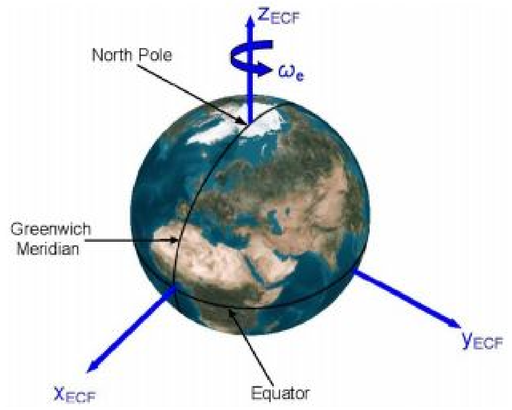

Figure 2.2: ECEF Coordinate System[3]

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.

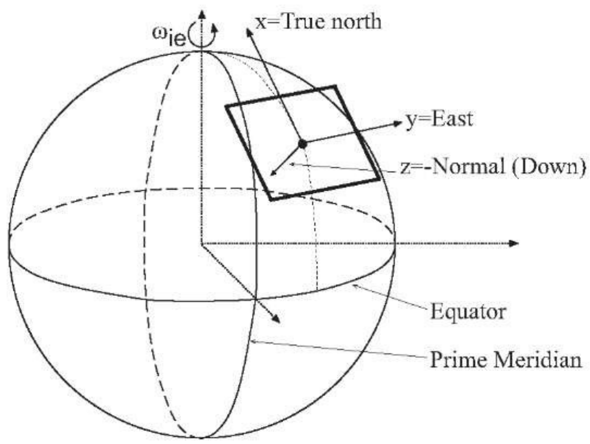

Figure 2.3: NED Coordinate System[4]

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.

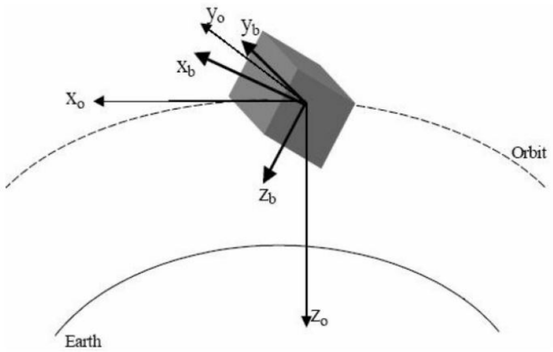

Figure 2.4: Orbit Reference frame and Body Coordinate System[2]

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:

$$v^y = R_x^y \times v^x \tag{2.1}$$

2.2.2 Euler Angles

The successive angular rotations about the three orthogonal body axes are defined as the Euler angle rotations[2]. Roll(ϕ), pitch(θ) and yaw(ψ) 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,

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & sin\phi \\ 0 & -sin\phi & cos\phi \end{bmatrix} \tag{2.2}$$

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.

$$R_y(\theta) = \begin{bmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix} \tag{2.3}$$

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.

$$R_z(\psi) = \begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

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:

$$\underline{q} = q_1\mathbf{i} + q_2\mathbf{j} + q_3\mathbf{k} + q_4 \tag{2.5}$$

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 $\underline{q_{1-3}}$ [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].

$$q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1 \tag{2.6}$$

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*.

$$\underline{q}_{31} = \underline{q}_{34}\underline{q}_{41} \tag{2.7}$$

During the pointing control modes of space craft, the attitude error between the reference attitude quaternion and the current attitude quaternion is also

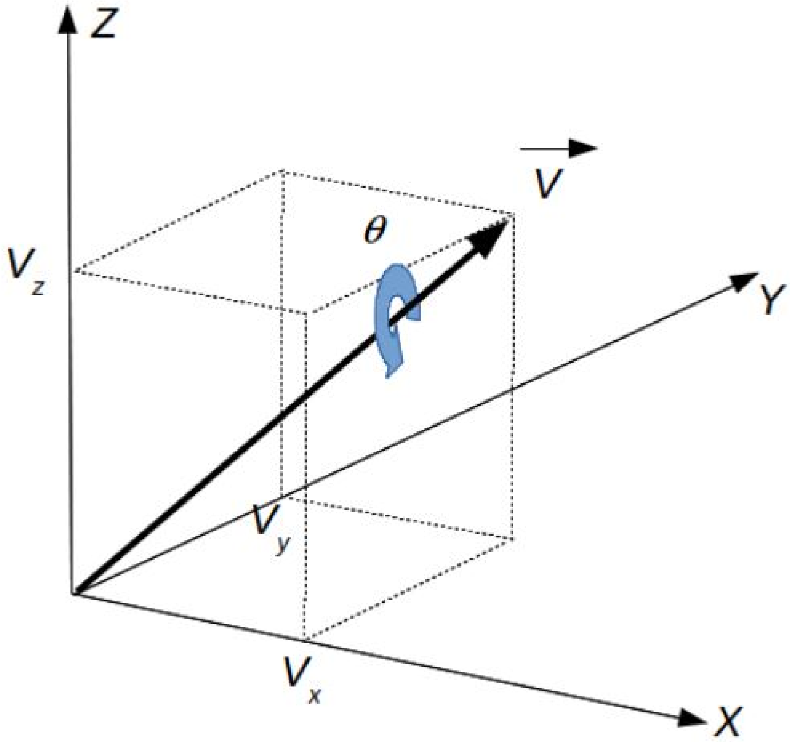

Figure 2.5: Quaternion Representation of rotation [6]

calculated in the similar way.

$$\underline{q}_{error} = \underline{q}_{ref}\underline{q}_{attitude} \tag{2.8}$$

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 conatins 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 space craft 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 space craft 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.

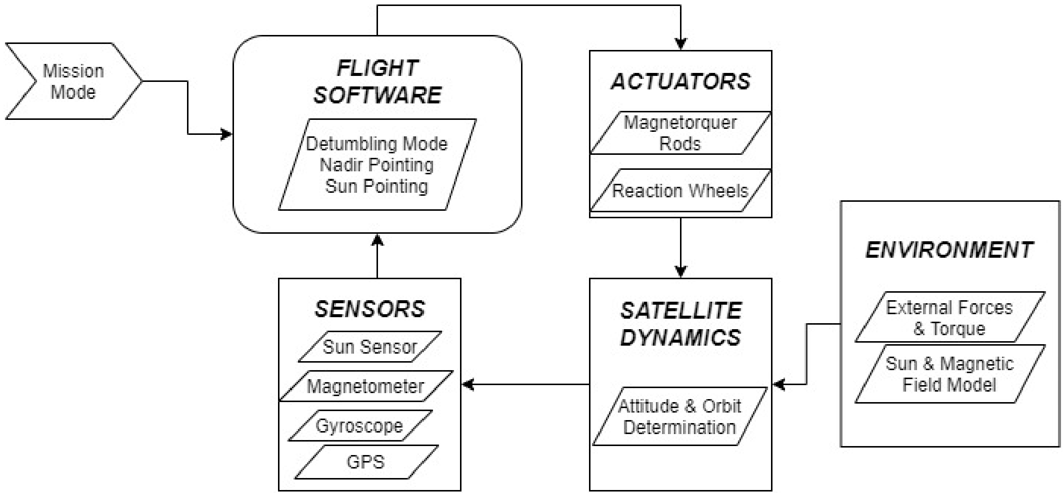

Figure 3.1: Attitude Control System Block Diagram

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 modes 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:

Keplerian Orbital Elements	**Values**
Semi-major Axis (a)	6786233.13
Eccentricity (e)	0.0010537
Inclination (i)	51.7519
Right Ascension of Ascending Node (Ω)	95.2562
Argument of periapsis (ω)	93.4872
True Anomaly (v)	302.9234
Epoch Date & Time	4-January-2019-12:00

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:

$$F = G\frac{Mm}{r^2} \tag{3.1}$$

where **G** is the gravitational constant and its values 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.

$$F = ma \tag{3.2}$$

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

Input	Unit	Output	Unit
xECEF (ECEF Position)	meters	EnvForceBody (Environmental Force on Body)	Newton

Table 3.2: Gravitational Model Simulink Block Inputs & Outputs

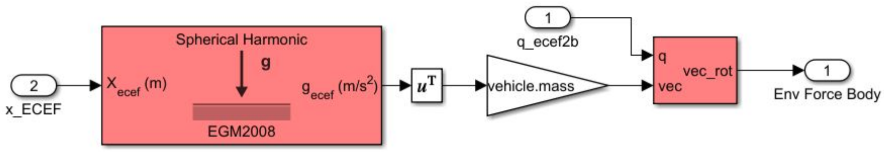

Figure 3.2: Gravitational Forces acting on the Satellite

the current timestamp in Julian Date format. The output is the sun vector calculated in the ECI frame of reference.
The Simulink block of *Sun Position Model's* inputs and outputs are given below.

3.2.3 Magnetic Field Model

The magnetic field model is also the part of aerospace block-set environment. It implements the 12^{th} 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

Input	Unit	Output	Unit
JD (Julian Date converted from Simulation time)	sec	xSumECI (Sun Vector in ECI Frame)	meters

Table 3.3: Sun Position Model Simulink Block Inputs & Outputs

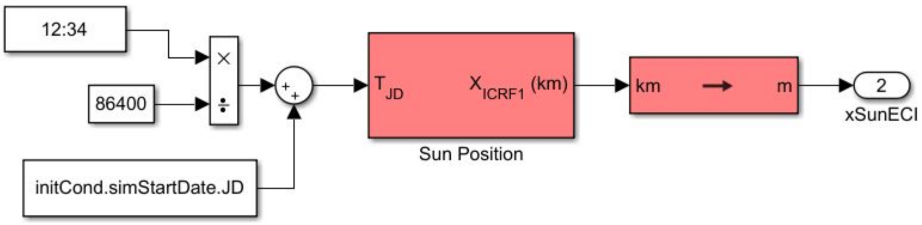

Figure 3.3: Sun Vector Model

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]:

$$T_G = (F_2 - F_1)\frac{l}{2}sin\alpha \tag{3.3}$$

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]:

$$F_{AD} = \frac{\rho C_D A V_o^2}{2} \tag{3.4}$$

Input	Unit	Output	Unit
Lat-Lon-Alt (Latitude, Longitude and Altitude of satellite)	deg, deg & meter	Bned (Magnetic Field Intensity Vector in NED Frame	Tesla

Table 3.4: Magnetic Field Model Simulink Block Inputs & Outputs

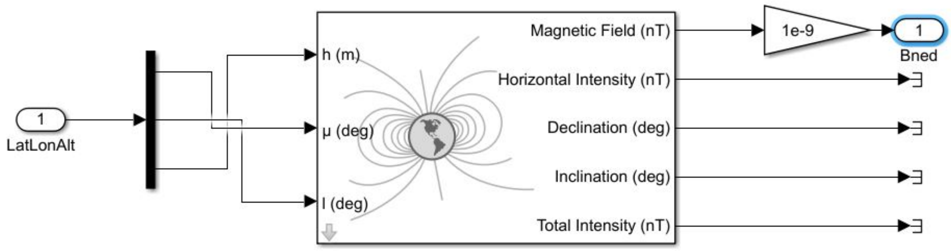

Figure 3.4: Magnetic Field Model

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.

$$\vec{T_m} = \vec{m}\vec{B} \tag{3.5}$$

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.

$$\vec{T_s} = \vec{r_{sp}} \times \vec{F_s}\vec{F_s} = (1+K)\rho_s A_\perp \tag{3.6}$$

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 space craft 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

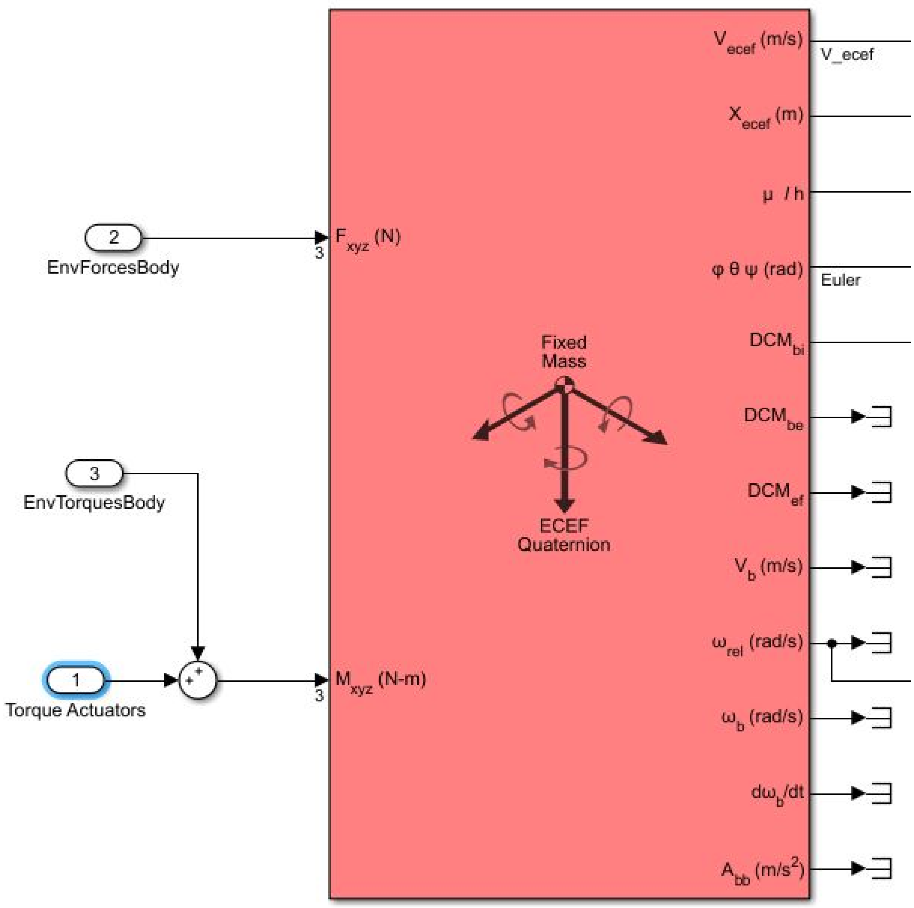

Figure 3.5: Satellite Dynamics Simulink Library Block

the sensors and after few transformations, the quaternions representing the attitude of space craft 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:

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

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.

Input	Unit	Output	Unit
MagEarth (3 $\times$ 1 Magnetic Field Intensity Vector in NED Frame)	Tesla	B_{MGM} (3$\times$1 Magnetic Field Intensity Vector in Body Frame)	Tesla
NED2Body (3$\times$3 NED to Body transformation Vector)	-		

Table 3.6: Magnetometer Simulink Block Inputs and Outputs

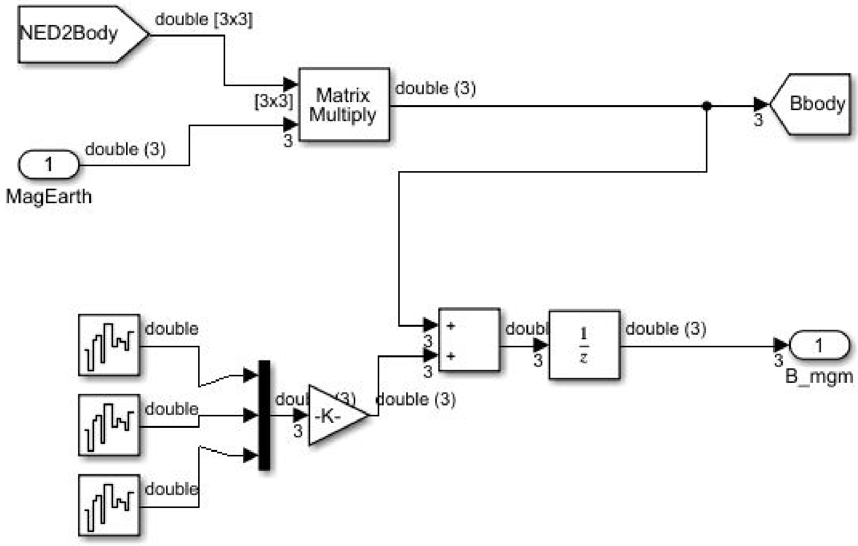

Figure 3.6: Magnetometer Simulink Block

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 (ω_{rel}).

Input	Unit	Output	Unit
ω (3 × 1 Angular velocity in body axes of satellite)	rad/sec (radians per second)	ω_{SENS} (3 × 1 Angular velocity in body axes affected by noises)	rad/sec (radians per second)

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

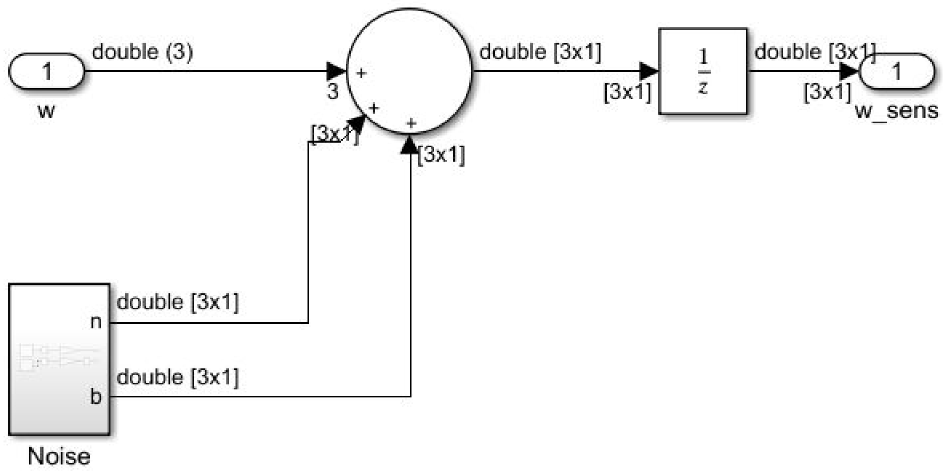

Figure 3.7: Gyroscope representation of Simulink Block

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.

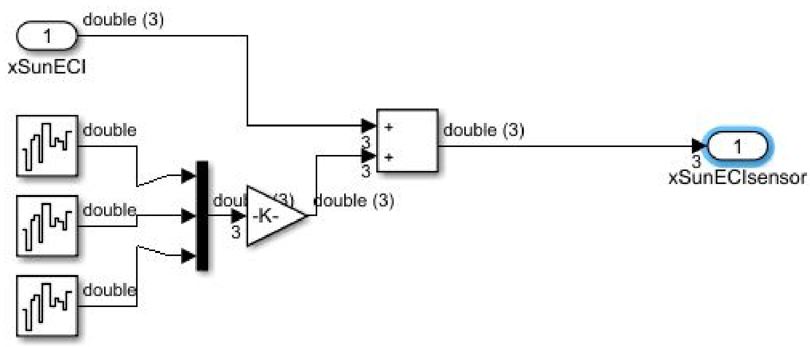

Figure 3.8: Sun Sensor representation of Simuink Block

Input	Unit	Output	Unit
xSunECI (3 × 1 Sun Position Vector in ECI Frame)	m (meters)	xSunECISensor (3 × 1 Sun Position Vector in ECI Frame affected by noises)	m (meters)

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.

Input	Unit	Output	Unit
xECEF (3×1 Position Vector in ECEF Frame)	m (meters)	xECEFGPS (3 × 1 Position Vector in ECEF Frame affected by noises)	m (meters)
vECEF (3×1 Velocity Vector in ECEF Frame)	m/s (meters per second)	vECEFGPS (3 × 1 Velocity Vector in ECEF Frame affected by noises)	m/s (meters per second)

Table 3.9: GPS Simulink Block Inputs & Outputs

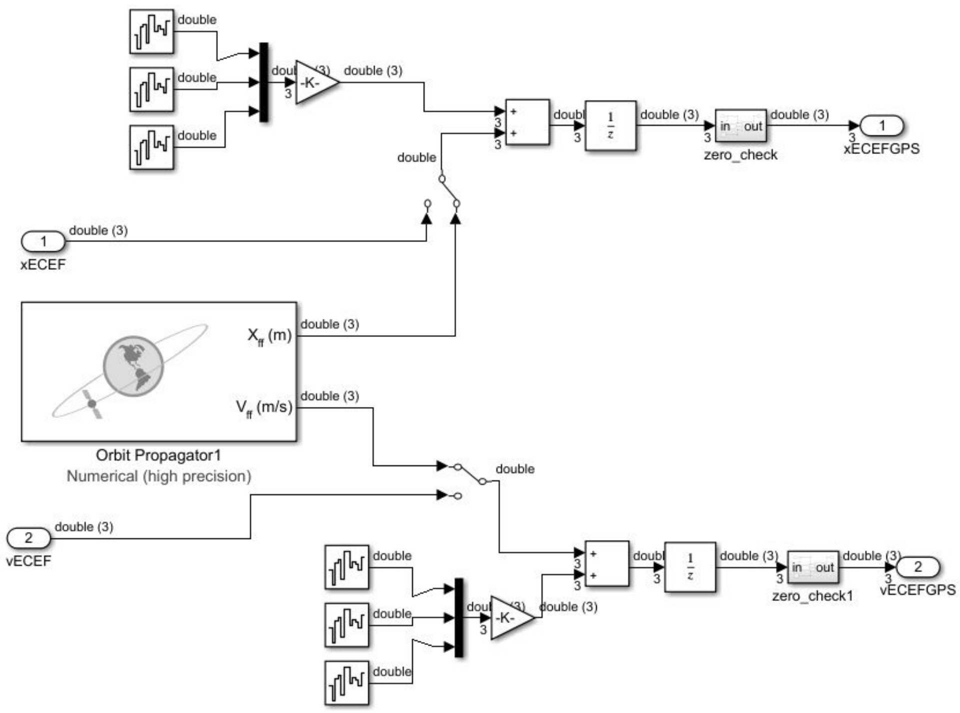

Figure 3.9: GPS representation of Simuink Block

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.

$$\vec{M} = NI\vec{A} \tag{3.7}$$

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.

$$\vec{\tau} = \vec{M} \times \vec{B} \tag{3.8}$$

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 genereated by the torquer rods which are mapped on each of the magnetorquer rod. The magnetic field vector in Body frame B_{Body} is multiplied with the alignment matrix of each magnetorquer rod to compute the torque generated by each rod in body axes.

Input	Unit	Output	Unit
DipoleCMD (3 × 1 Dipole Commands for each of three magnetorquer rods)	$A - m^2$ (Ampere-square-meter)	Torque (3 × 1 Torque vector applied on each of body axis)	N-m (Newton-meter)
B_{BODY} (3×1 Magnetic Field Intensity vector in Body frame)	Tesla		

Table 3.10: Magnetorquer Rods Simulink Block Inputs & Outputs

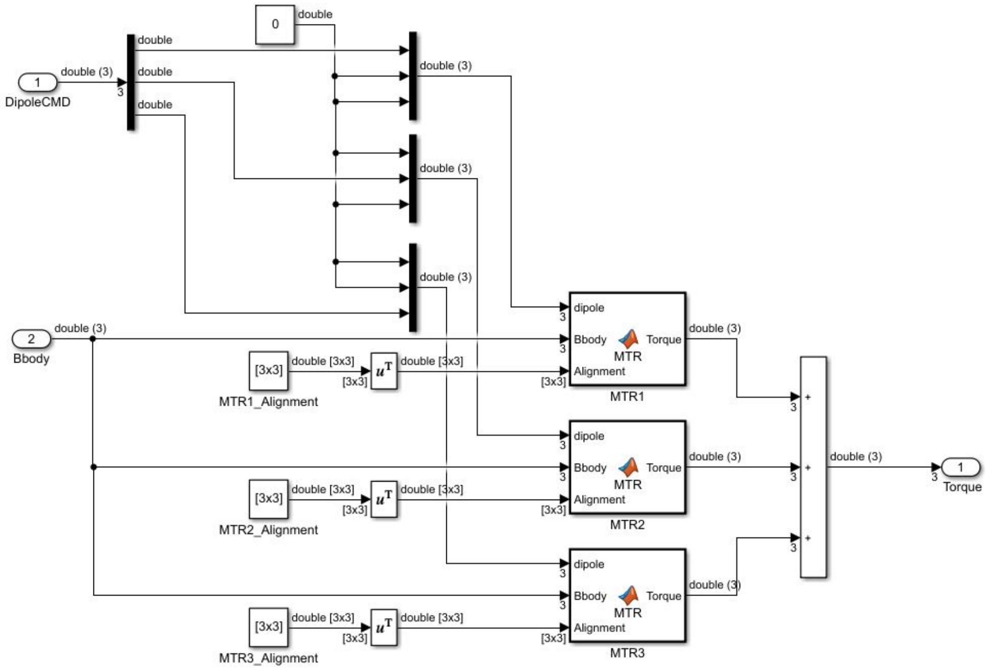

Figure 3.10: Magnetorquer Rods model Simulink representation

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[2]. The tetrahedron configuration is shown in figure. The homogeneous distribution matrix of Reaction wheels in tetrahedron configuration is shown.

$$L = \begin{bmatrix} \sqrt{3}/3 & -\sqrt{3}/3 & -\sqrt{3}/3 & \sqrt{3}/3 \\ \sqrt{3}/3 & -\sqrt{3}/3 & \sqrt{3}/3 & -\sqrt{3}/3 \\ \sqrt{3}/3 & \sqrt{3}/3 & -\sqrt{3}/3 & -\sqrt{3}/3 \end{bmatrix} \tag{3.9}$$

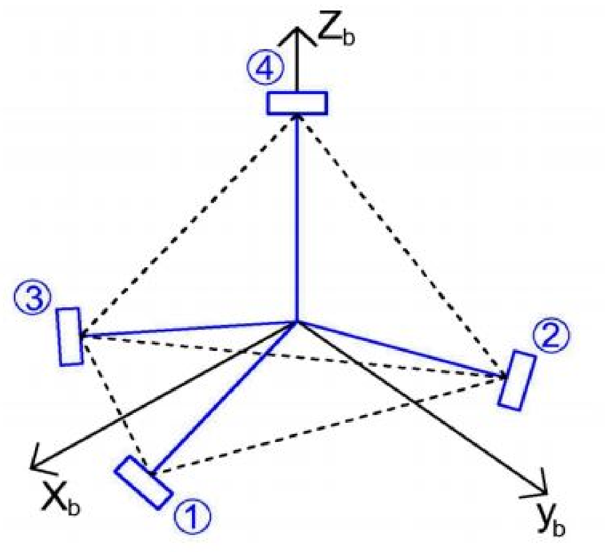

Figure 3.11: Tetrahedron configuration of Reaction Wheels[2]

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 multiplyin with distribution matrix. It is also assumed in this study, that the maximum torque that each wheel can produce is $5mN - m$, 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 farme.

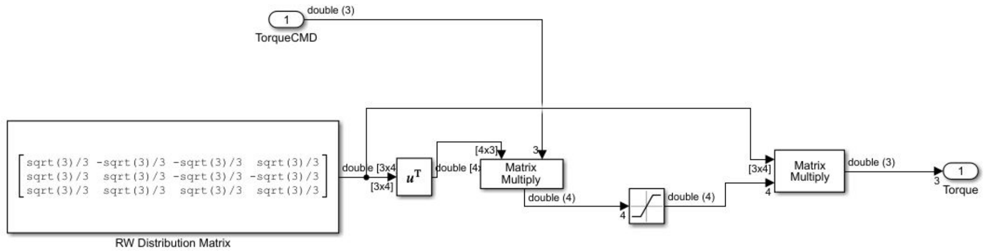

Figure 3.12: Reaction wheels representation of Simulink Model

Input	Unit	Output	Unit
TorqueCMD (3 × 1 Torque Commands required to be applied on body axes)	$N-m$ (Newton-meter)	Torque (3 × 1 Torque vector applied on each of body axis)	$N-m$ (Newton-meter)

Table 3.11: Reaction wheels Simulink Block Inputs & Outputs

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

- 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.

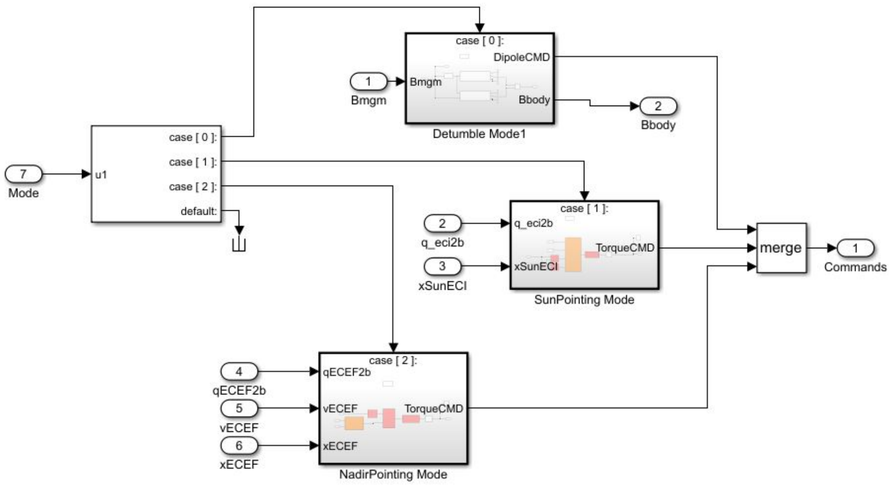

Figure 3.13: Flight Software Top Level hierarchy in Simulink

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:

$$M_{bdot} = -M_{max}.sign(\dot{B}_b) \tag{3.10}$$

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] :

$$M_{bdot} = -k_b.\frac{\dot{B}_b}{||B_b||} \tag{3.11}$$

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

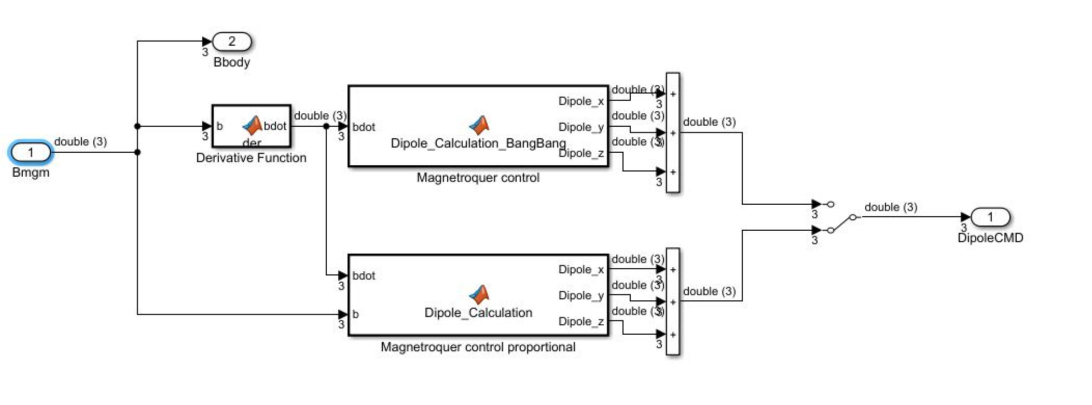

Figure 3.14: Detumbling Mode Matlab/Simulink Block

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

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.

$$FirstAlignmentVector = [0 \quad 0 \quad -1] \tag{3.12}$$

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.

$$SecondAlignmentVector = [1 \quad 0 \quad 0] \tag{3.13}$$

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:

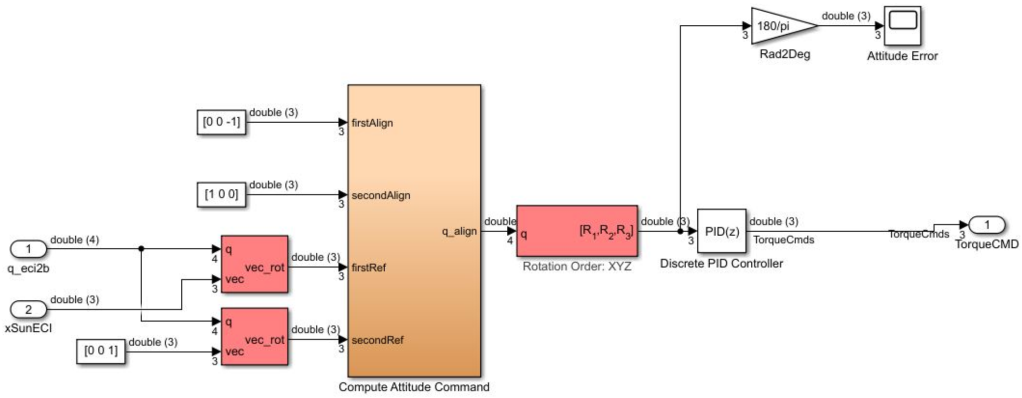

Figure 3.15: Sun Pointing Mode Simulink Model

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

Input	Unit	Output	Unit
xSunECI (3 × 1 Sun position Vector in ECI frame)	m meters	TorqueCMD (3 × 1 Torque Command vector in each of body axis)	$N - m$ (Newton-meter)
qECI2b (4 × 1 Quaternions between Body frame to ECI frame)	-		

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

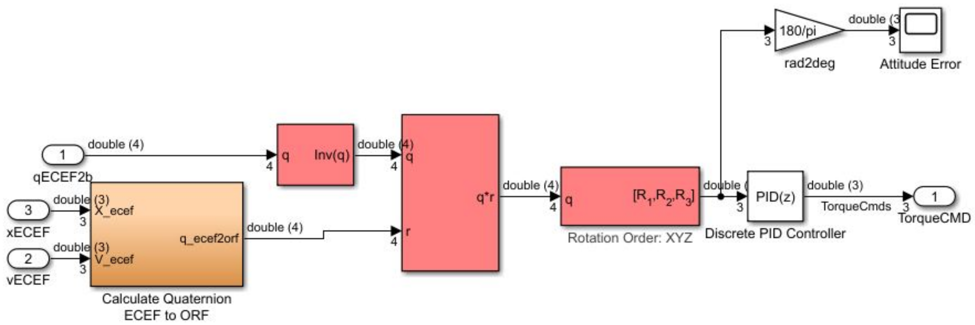

Figure 3.16: Nadir Pointing Mode Simulink Model

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

Input	Unit	Output	Unit
xECEF (3 × 1 Satellite Position Vector in ECEF frame)	m meters	TorqueCMD (3 × 1 Torque Command vector in each of body axis)	$N-m$ (Newton-meter)
vECEF (3 × 1 Satellite Velocity Vector in ECEF frame)	m/s meters per second		
qECEF2b (4 × 1 Quaternions between Body frame to ECEF frame)	-		

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.

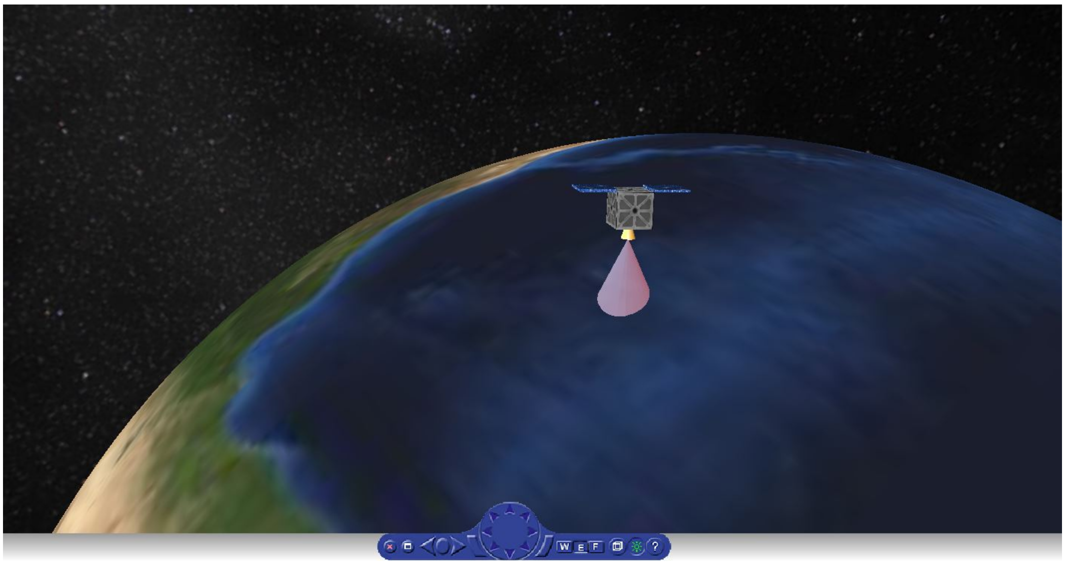

Figure 3.17: Three-Dimensional Visualization of satellite in Orbit

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 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:

Data Structure	Description
ExtUVehicleT	External Inputs
ExtYVehicleT	External Outputs
ConstBVehicleT	Constants Data structure
DWVehicleT	States variables Data Structure
BVehicleT	Block Signals which are defined for intermediate operations on data

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.

- **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:

$$HardenedData :: TripleData < Datatype > Variable; \tag{4.1}$$

The three copies of data can be accessed as:

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

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 *TripleData.h* file.

There is also another file, *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 *radiation hardened* or *unhardened* software. The example of variable declaration is shown:

$$HardenedData :: SingleData < Datatype > Variable; \tag{4.2}$$

The data can be accessed as:

Variable.a();

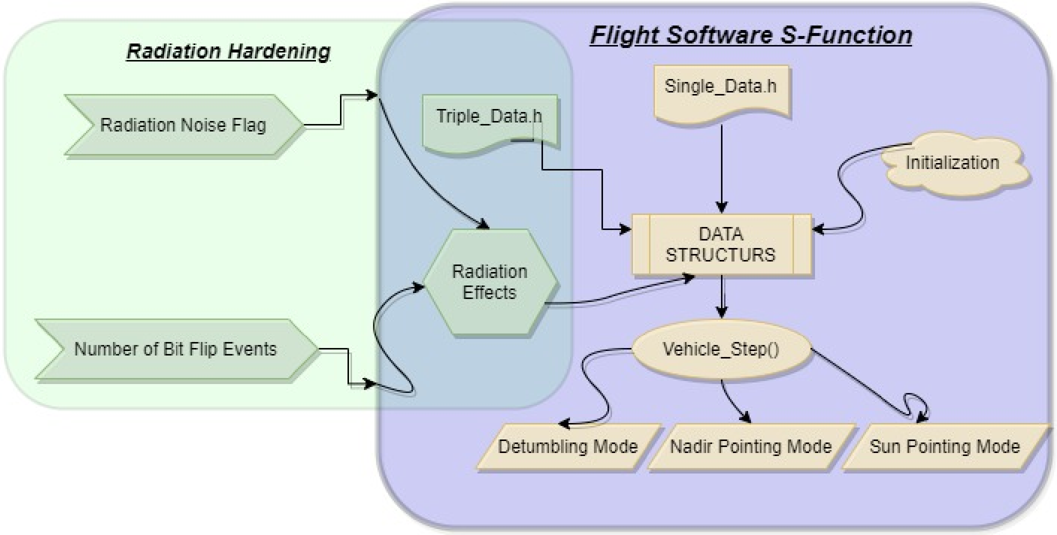

Figure 4.1: Radiation Hardened Software Hierarchy

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:

Variable.a() = 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.

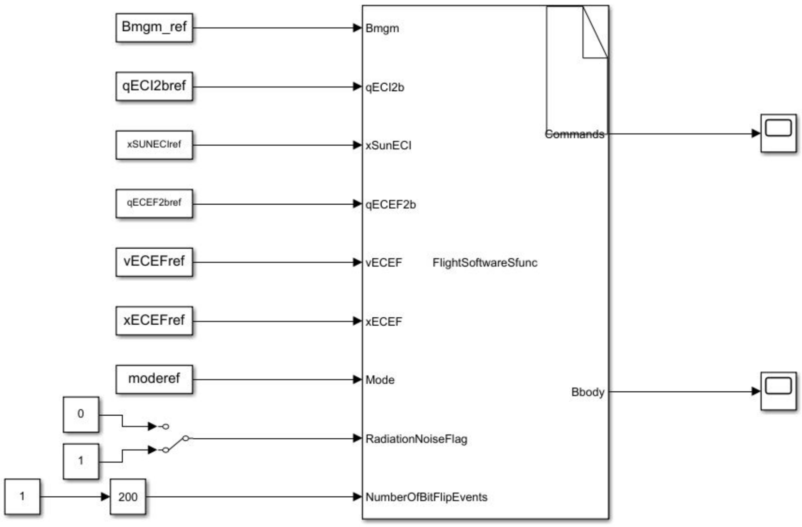

Figure 4.2: S-function Builder Representation of Flight Software

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 seperated 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 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 *bdot* 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:

$$\omega_b = [5 \quad 4 \quad -0.0517] \qquad deg/sec \tag{5.1}$$

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 $0.5 deg/sec$. The control law took almost 6 hours to damp the rates.

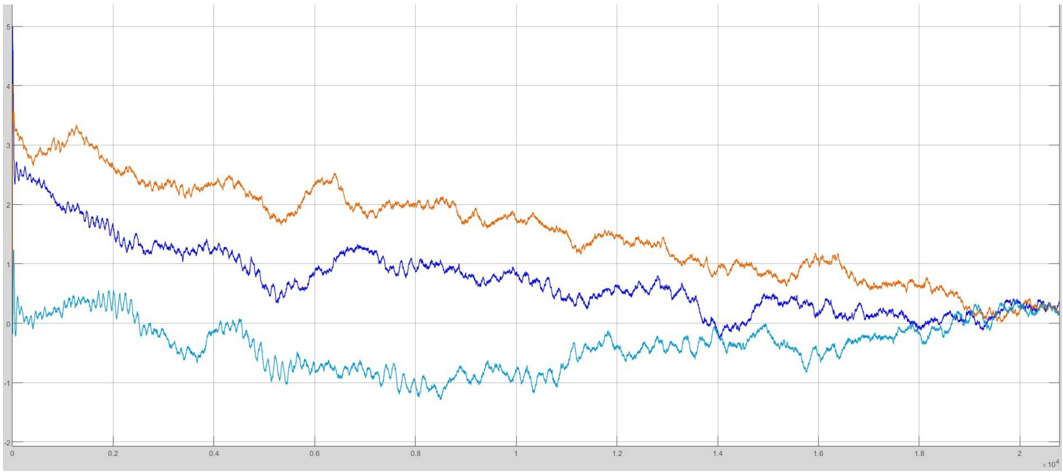

Figure 5.1: Detumbling Mode Simulation Results: Angular rates along xyz body axes

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.

The angular rates at the start of this mode in the body are considered as $0.4deg/sec$ in x and y axes, and $-0.0517deg/sec$ 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.

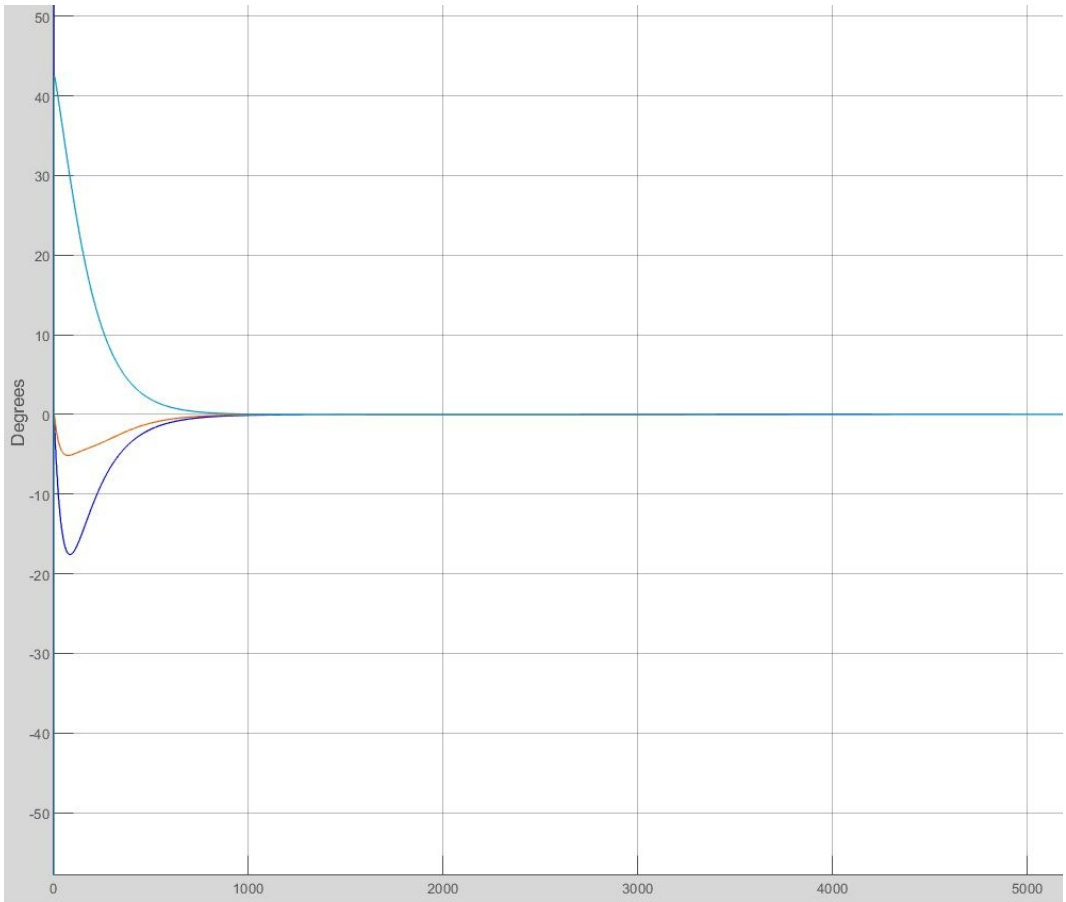

Figure 5.2: Nadir Pointing Mode Attitude Error: Euler angles

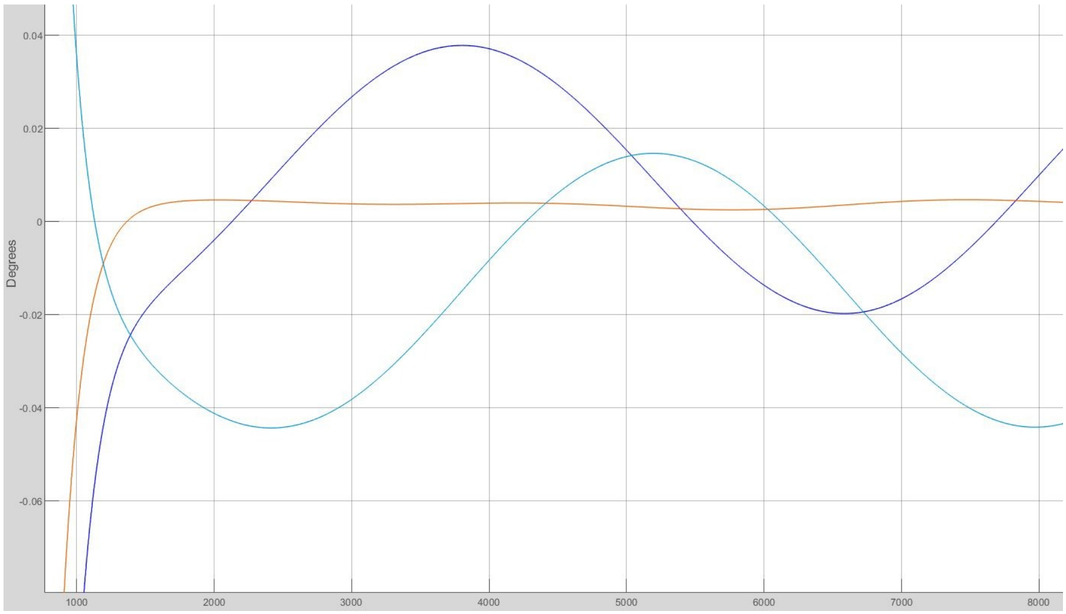

Figure 5.3: Nadir Pointing Mode Attitude Error steady state: Euler angles

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

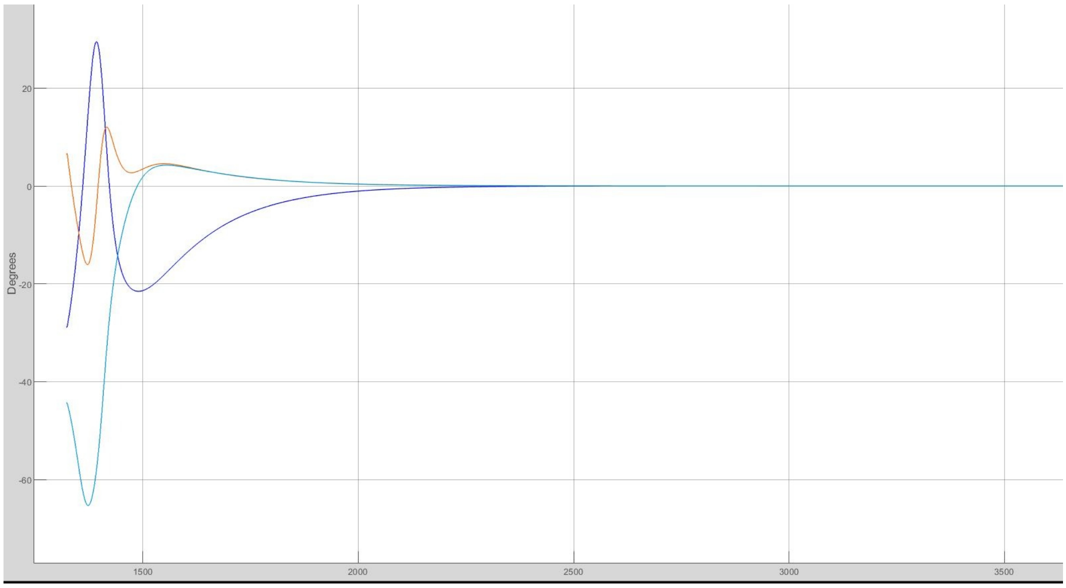

Figure 5.4: Sun Pointing Mode Attitude Error: Euler angles

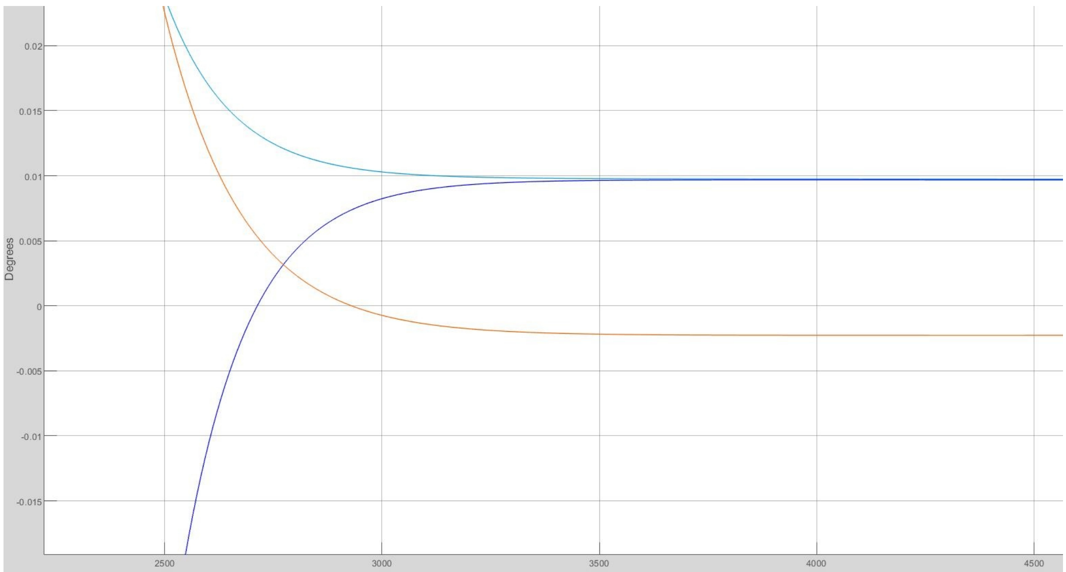

Figure 5.5: Sun Pointing Mode Attitude Error steady state: Euler angles

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_{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_{Body} are in the range of -5×10^{-5} to 5×10^{-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.

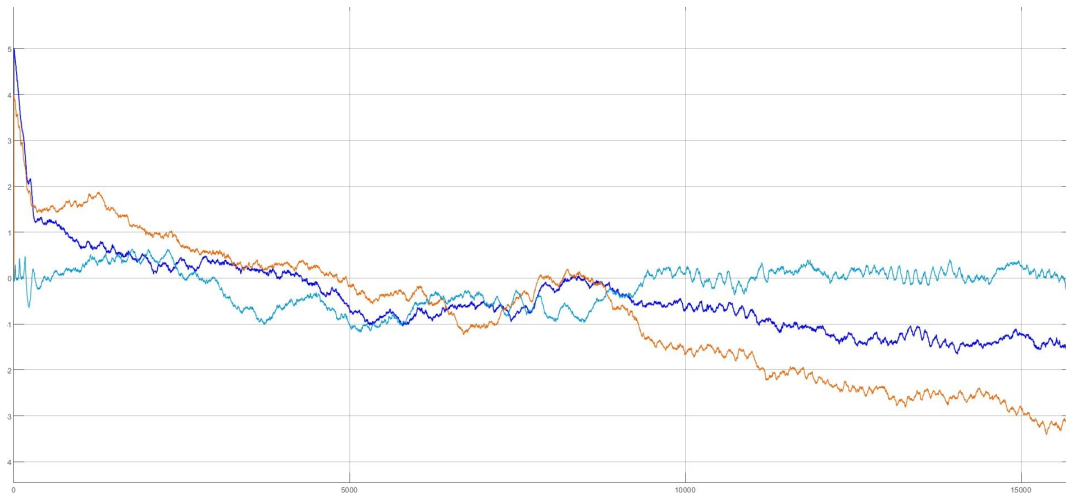

Figure 5.6: Radiations affected Detumbling Mode Simulation Results: Angular rates along xyz body axes

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 *Radiation Noise Flag* is set to 1, the attitude error constantly changes to different values. Therefore, it is impossible to achieve the required attitude.

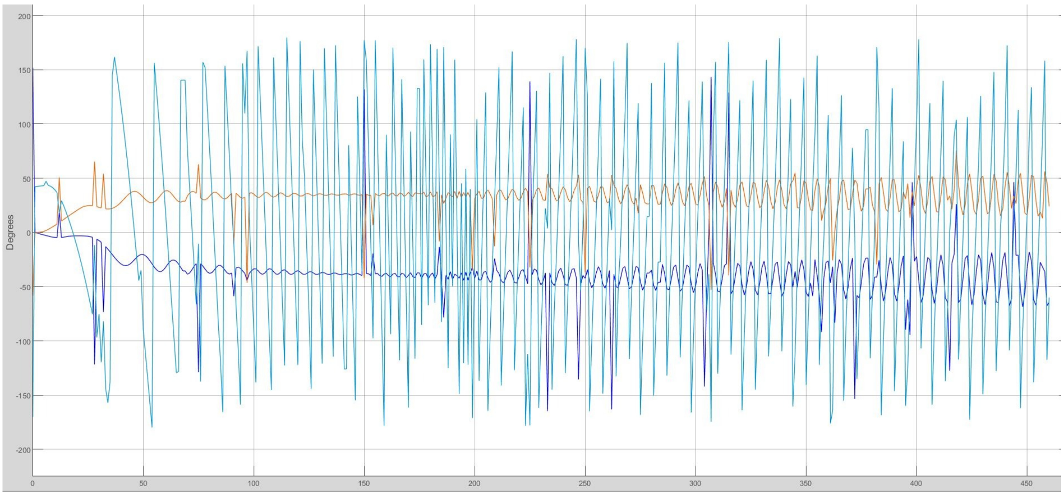

Figure 5.7: Radiations affected Nadir Pointing Mode Simulation Results: Euler angles

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 *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.

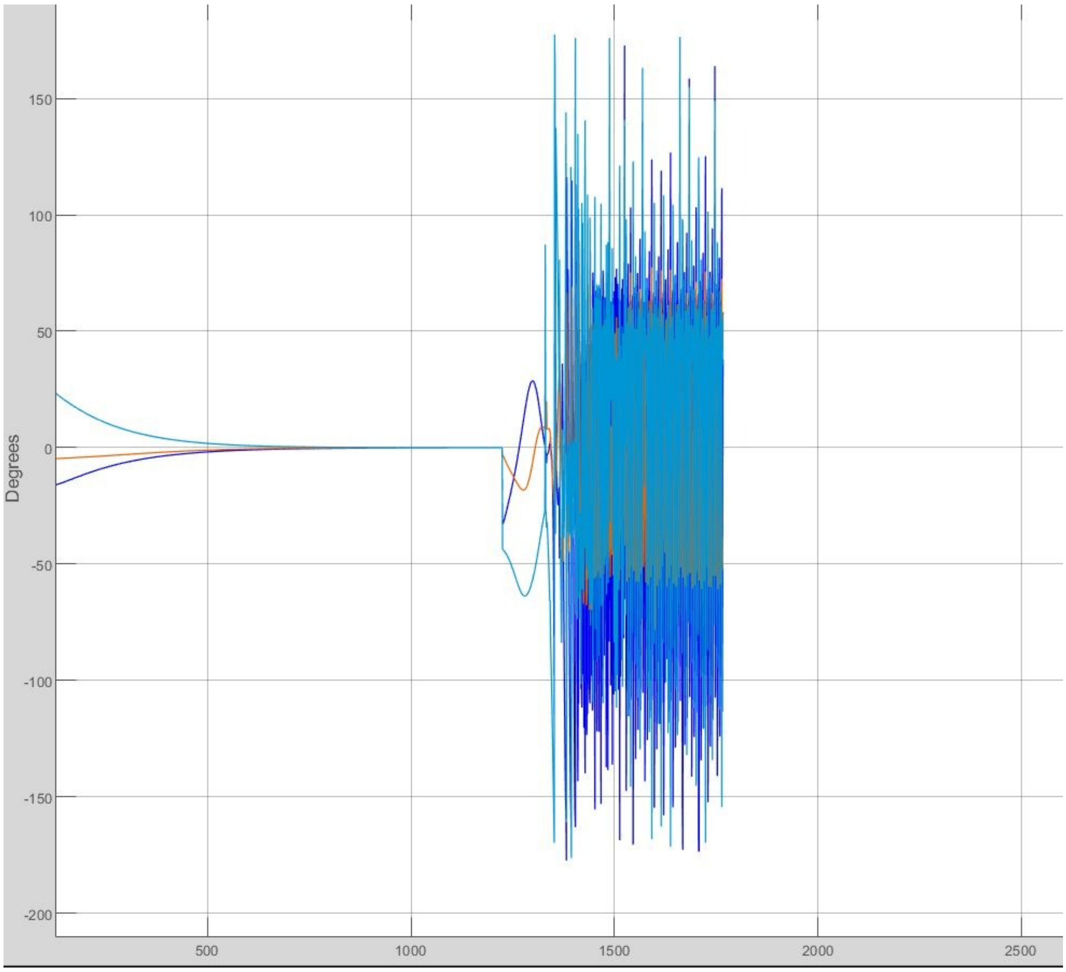

Figure 5.8: Radiations affected Sun Pointing Mode Simulation Results: Euler angles

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.

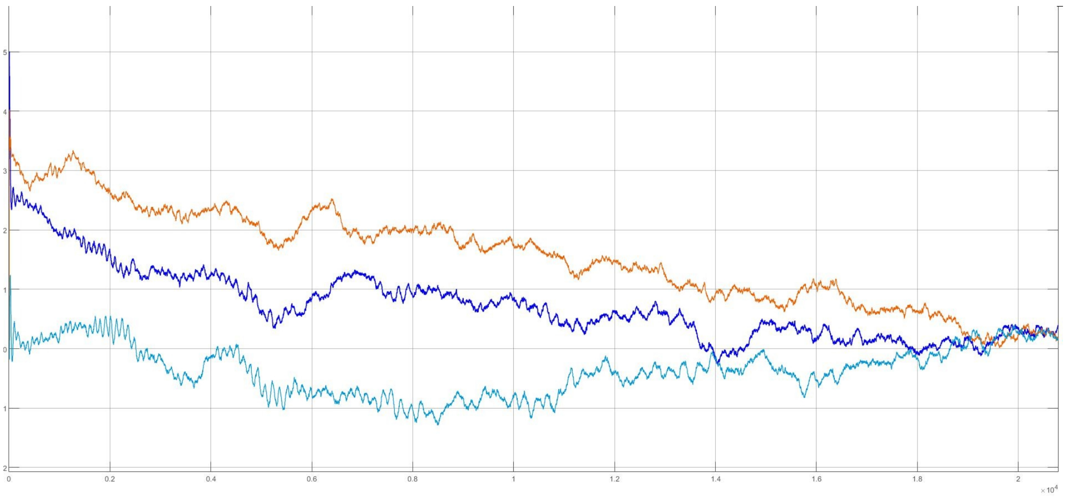

Figure 5.9: Radiation Hardened Detumbling Mode Simulation Results: Angular rates along xyz body axes

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.

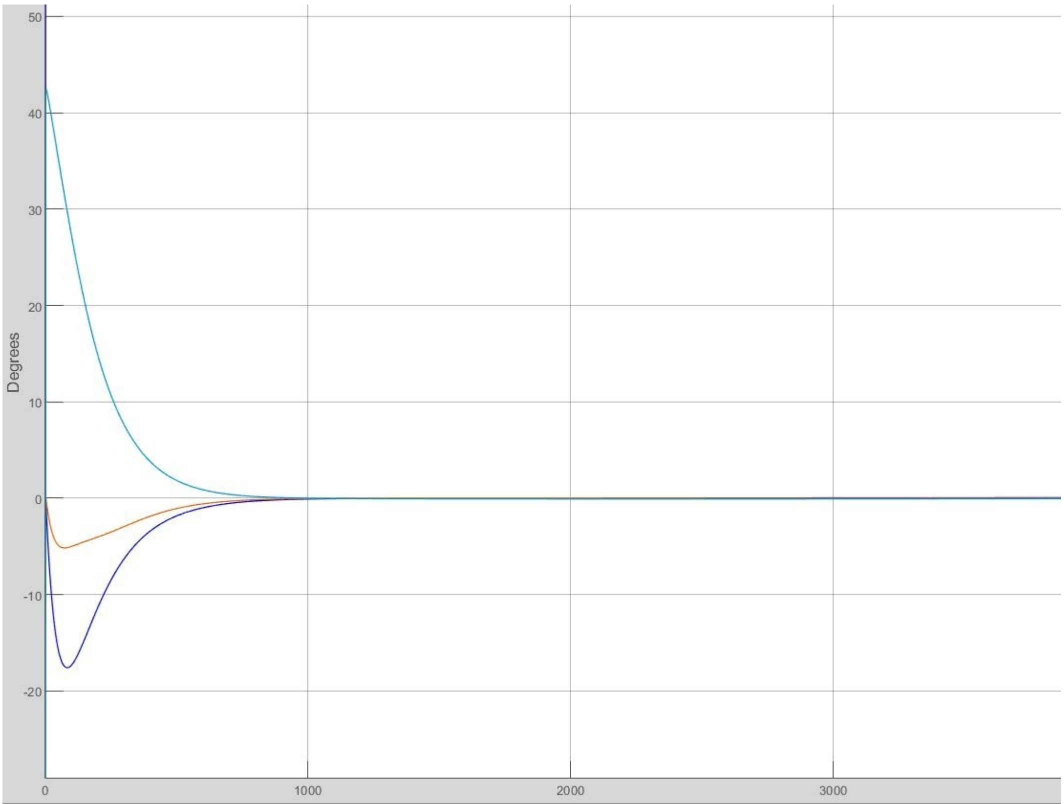

Figure 5.10: Radiation Hardened Nadir Pointing Mode Simulation Results: Euler angles

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.

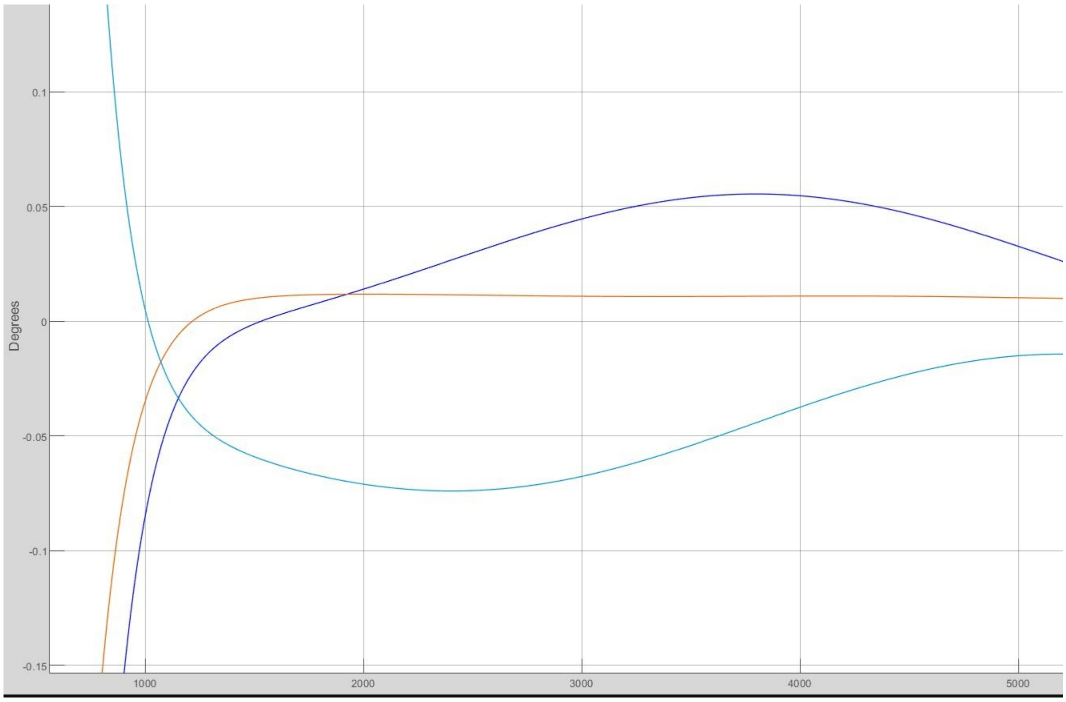

Figure 5.11: Radiation Hardened Nadir Pointing Mode steady state Simulation Results: Euler angles

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.

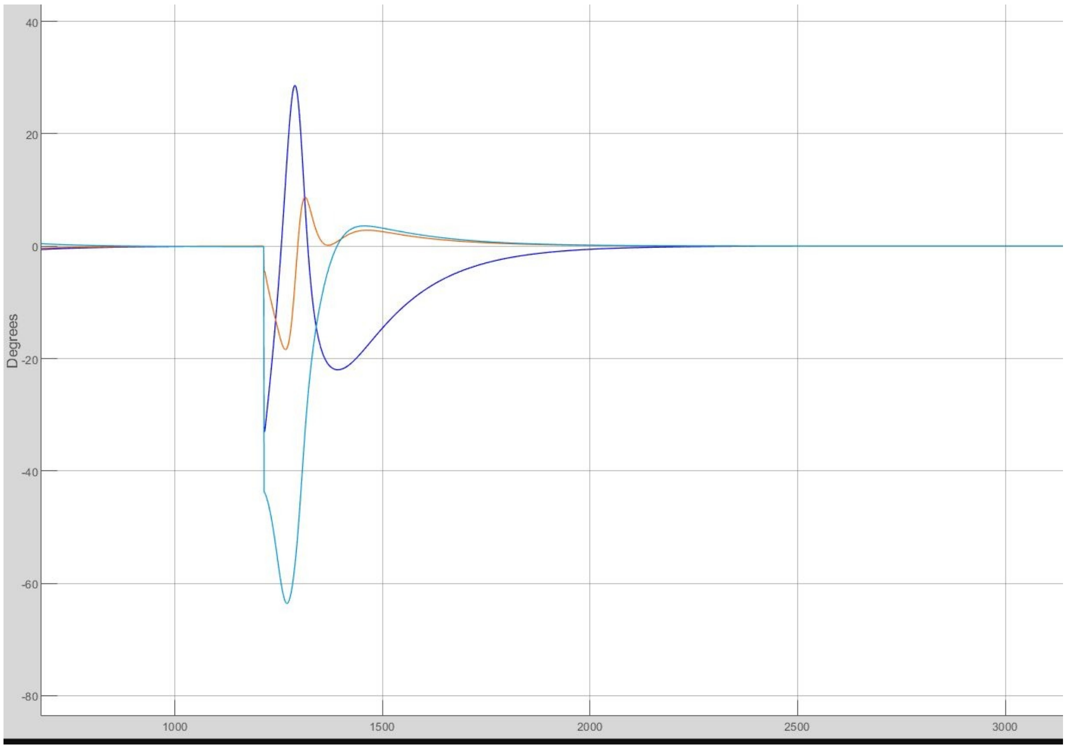

Figure 5.12: Radiation Hardened Sun Pointing Mode Simulation Results: Euler angles

Detumbling Mode	**Damped Rates**	**Comments**
Simulink Model	0.5 deg/sec	It took 6 hours to damp the rates withing threshold
Radiations Affected s-function	*Un-damped*	The angular rates of satellite remains constant around 3 deg/sec even after 6 hours
Radiations Hardened s-function	0.5 deg/sec	After 6 hours, the rates are damped within threshold

Table 5.1: Detumbling mode performance comparison

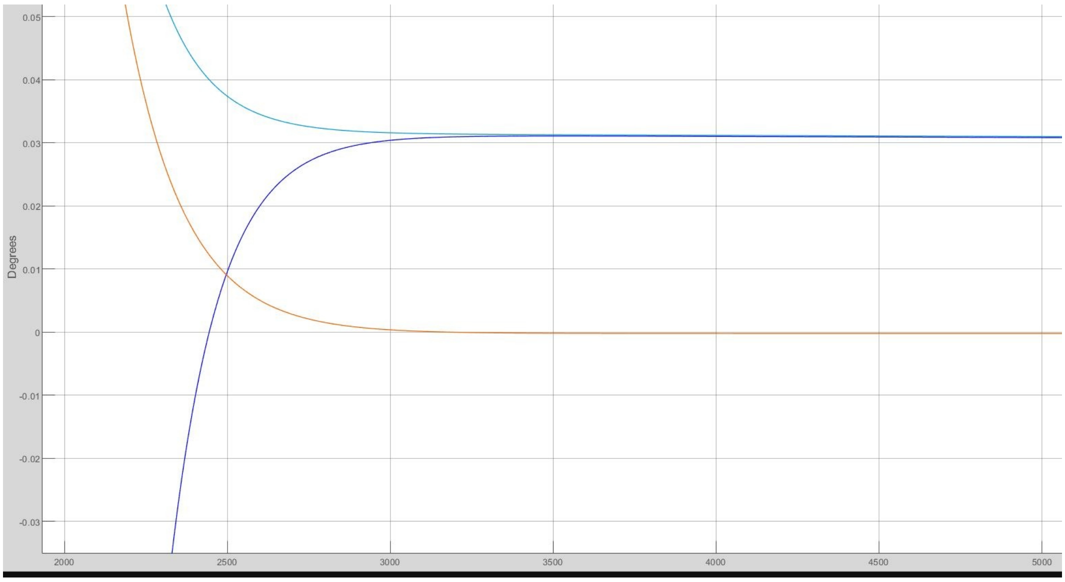

Figure 5.13: Radiation Hardened Sun Pointing Mode steady state Simulation Results: Euler angles

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

Table 5.2: Nadir Pointing mode performance comparison

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

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

```cpp
#include "Vehicle.h"


/*The following is the declarations of date structures which are used in the flight software*/

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

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

/* External inputs: The inputs of the flight software */
ExtU_Vehicle_T Vehicle_U;

/* External outputs: The outputs of the flight software */
ExtY_Vehicle_T Vehicle_Y;

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




/*The entry point of the flight software. Simulink s-function calls this subroutine to execute the flight software at each time stamp
 * Inputs: 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,unsigned int RadiationNoiseFlag, unsigned int NumberOfBitFlipEvents
 * Outputs: double Commands[3], double Bbody[3], double AttitudeError[3]
 * */
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;

    /* Initialization of structures need to be done only once*/
    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.Bmgm[2] = B_mgm[2];
      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.xSunECI[2] = xSunECI[2];
      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.vECEF[2] = vECEF[2];
      Vehicle_U.xECEF[0] = xECEF[0];
      Vehicle_U.xECEF[1] = xECEF[1];
      Vehicle_U.xECEF[2] = xECEF[2];
      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();

```

```
        /* external outputs Mapping*/
        Commands[0]=  (real_T)Vehicle_Y.Commands[0]  ;
        Commands[1]= (real_T) Vehicle_Y.Commands[1] ;
        Commands[2]= (real_T) Vehicle_Y.Commands[2] ;
        Bbody[0]= (real_T) Vehicle_Y.Bbody[0] ;
        Bbody[1]=  (real_T)Vehicle_Y.Bbody[1] ;
        Bbody[2]=  (real_T)Vehicle_Y.Bbody[2] ;
        AttitudeError[0] = (real_T)Vehicle_B.VectorConcatenate_a[0];
        AttitudeError[1] = (real_T)Vehicle_B.VectorConcatenate_a[1];
        AttitudeError[2] = (real_T)Vehicle_B.VectorConcatenate_a[2];
}
```

DetumblingMode.cpp

```
#include "Vehicle.h"

void Detumble()
{
      real_T tmpForInput_idx_0;

      /* Outputs for IfAction SubSystem: '<S1>/Detumble Mode' incorporates:
         *  ActionPort: '<S2>/Action Port'
         */
        /* Outport: '<Root>/Bbody' incorporates:
         *  Inport: '<Root>/Bmgm'
         *  Inport: '<S2>/Bmgm'
         */
        /* MATLAB Function 'Vehicle Flight Software/Detumble Mode/Derivative Function': '<S5>:1' */
        /* '<S5>:1:4' if isempty(state) */
        /* '<S5>:1:7' bdot = b-state; */
        /* '<S5>:1:8' state = b; */
        Vehicle_Y.Bbody[0] = Vehicle_U.Bmgm[0];

        /* MATLAB Function: '<S2>/Derivative Function' incorporates:
         *  Outport: '<Root>/Bbody'
         */
        Vehicle_B.bdot[0] = Vehicle_Y.Bbody[0] - Vehicle_DW.state[0];
        Vehicle_DW.state[0] = Vehicle_Y.Bbody[0];

        /* Outport: '<Root>/Bbody' incorporates:
         *  Inport: '<Root>/Bmgm'
         *  Inport: '<S2>/Bmgm'
         */
        Vehicle_Y.Bbody[1] = Vehicle_U.Bmgm[1];

        /* MATLAB Function: '<S2>/Derivative Function' incorporates:
         *  Outport: '<Root>/Bbody'
         */
        Vehicle_B.bdot[1] = Vehicle_Y.Bbody[1] - Vehicle_DW.state[1];
        Vehicle_DW.state[1] = Vehicle_Y.Bbody[1];

        /* Outport: '<Root>/Bbody' incorporates:
         *  Inport: '<Root>/Bmgm'
         *  Inport: '<S2>/Bmgm'
         */
        Vehicle_Y.Bbody[2] = Vehicle_U.Bmgm[2];

        /* MATLAB Function: '<S2>/Derivative Function' incorporates:
         *  Outport: '<Root>/Bbody'
         */
        Vehicle_B.bdot[2] = Vehicle_Y.Bbody[2] - Vehicle_DW.state[2];
        Vehicle_DW.state[2] = Vehicle_Y.Bbody[2];

        /* MATLAB Function: '<S2>/Magnetroquer control' */
        /* MATLAB Function 'Vehicle Flight Software/Detumble Mode/Magnetroquer control': '<S6>:1' */
        /* '<S6>:1:3' mmax=6; */
        /* '<S6>: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 = 1.732050807568877e-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>:1:7' 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[1] = -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>:1:8' Dipole_z= [0;0;-mmax*sign(bdot(3))]; */
```

```
        Vehicle_B.Dipole_z[0] = 0.0;
        Vehicle_B.Dipole_z[1] = 0.0;
    /*    tmpForInput_idx_0 = (real_T)Vehicle_B.bdot[2];
        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_z[2] = -6.0 * tmpForInput_idx_0;
        Vehicle_B.Dipole_z[2] = -1024.0 * Vehicle_B.bdot[2]/Vehicle_B.norm_b;

        /* End of MATLAB Function: '<S2>/Magnetroquer control' */

        /* Merge: '<S1>/Merge' incorporates:
         *  Sum: '<S2>/Add'
         */
        Vehicle_B.Merge[0] = (Vehicle_B.Dipole_x[0] + Vehicle_B.Dipole_y[0]) +
          Vehicle_B.Dipole_z[0];
        Vehicle_B.Merge[1] = (Vehicle_B.Dipole_x[1] + Vehicle_B.Dipole_y[1]) +
          Vehicle_B.Dipole_z[1];
        Vehicle_B.Merge[2] = (Vehicle_B.Dipole_x[2] + Vehicle_B.Dipole_y[2]) +
          Vehicle_B.Dipole_z[2];

        /* End of Outputs for SubSystem: '<S1>/Detumble Mode' */

}
```

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 IfAction SubSystem: '<S1>/NadirPointing Mode' incorporates:
         *  ActionPort: '<S3>/Action Port'
         */
        /* Product: '<S20>/j x k' incorporates:
         *  Inport: '<Root>/vECEF'
         *  Inport: '<Root>/xECEF'
         */
        Vehicle_B.jxk = Vehicle_U.vECEF[1] * Vehicle_U.xECEF[2];

        /* Product: '<S20>/k x i' incorporates:
         *  Inport: '<Root>/vECEF'
         *  Inport: '<Root>/xECEF'
         */
        Vehicle_B.kxi = Vehicle_U.vECEF[2] * Vehicle_U.xECEF[0];

        /* Product: '<S20>/i x j' incorporates:
         *  Inport: '<Root>/vECEF'
         *  Inport: '<Root>/xECEF'
         */
        Vehicle_B.ixj = Vehicle_U.vECEF[0] * Vehicle_U.xECEF[1];

        /* Product: '<S21>/k x j' incorporates:
         *  Inport: '<Root>/vECEF'
         *  Inport: '<Root>/xECEF'
         */
        Vehicle_B.kxj = Vehicle_U.vECEF[2] * Vehicle_U.xECEF[1];

        /* Product: '<S21>/i x k' incorporates:
         *  Inport: '<Root>/vECEF'
         *  Inport: '<Root>/xECEF'
         */
        Vehicle_B.ixk = Vehicle_U.vECEF[0] * Vehicle_U.xECEF[2];

        /* Product: '<S21>/j x i' incorporates:
         *  Inport: '<Root>/vECEF'
         *  Inport: '<Root>/xECEF'
         */
        Vehicle_B.jxi = 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.ixk;
        Vehicle_B.Sum[2] = Vehicle_B.ixj - Vehicle_B.jxi;

        /* Gain: '<S7>/z = -r' incorporates:
         *  Inport: '<Root>/xECEF'
         */
        Vehicle_B.zr[0] = -Vehicle_U.xECEF[0];
        Vehicle_B.zr[1] = -Vehicle_U.xECEF[1];
        Vehicle_B.zr[2] = -Vehicle_U.xECEF[2];

        /* Product: '<S18>/i x j' */
        Vehicle_B.ixj_f = Vehicle_B.Sum[0] * Vehicle_B.zr[1];

        /* Product: '<S18>/j x k' */
        Vehicle_B.jxk_o = Vehicle_B.Sum[1] * Vehicle_B.zr[2];

        /* Product: '<S18>/k x i' */
        Vehicle_B.kxi_e = Vehicle_B.Sum[2] * Vehicle_B.zr[0];

        /* Product: '<S19>/i x k' */
        Vehicle_B.ixk_g = Vehicle_B.Sum[0] * Vehicle_B.zr[2];
```

```
        /* Product: '<S19>/j x i' */
        Vehicle_B.jxi_l = Vehicle_B.Sum[1] * Vehicle_B.zr[0];

        /* Product: '<S19>/k x j' */
        Vehicle_B.kxj_a = Vehicle_B.Sum[2] * Vehicle_B.zr[1];

        /* Sum: '<S12>/Sum' */
        Vehicle_B.Sum_l[0] = Vehicle_B.jxk_o - Vehicle_B.kxj_a;
        Vehicle_B.Sum_l[1] = Vehicle_B.kxi_e - Vehicle_B.ixk_g;
        Vehicle_B.Sum_l[2] = Vehicle_B.ixj_f - Vehicle_B.jxi_l;

        /* DotProduct: '<S15>/Dot Product' */
        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;

        /* DotProduct: '<S15>/Dot Product' */
        Vehicle_B.DotProduct = tmpForInput_idx_2;

        /* Sum: '<S15>/Sum of Elements' */
        tmpForInput_idx_0 = Vehicle_B.DotProduct;

        /* Sum: '<S15>/Sum of Elements' */
        Vehicle_B.SumofElements = tmpForInput_idx_0;

        /* Math: '<S15>/Math Function1'
         *
         * About '<S15>/Math Function1':
         *  Operator: sqrt
         */
        tmpForInput_idx_0 = Vehicle_B.SumofElements;
        if (tmpForInput_idx_0 < 0.0) {
          /* Math: '<S15>/Math Function1'
           *
           * About '<S15>/Math Function1':
           *  Operator: sqrt
           */
          Vehicle_B.MathFunction1 = -std::sqrt(std::abs(tmpForInput_idx_0));
        } else {
          /* Math: '<S15>/Math Function1'
           *
           * About '<S15>/Math Function1':
           *  Operator: sqrt
           */
          Vehicle_B.MathFunction1 = std::sqrt(tmpForInput_idx_0);
        }

        /* End of Math: '<S15>/Math Function1' */

        /* Product: '<S15>/Divide' */
        Vehicle_B.ORFtoECEF[0] = Vehicle_B.Sum_l[0] / Vehicle_B.MathFunction1;

        /* DotProduct: '<S16>/Dot Product' */
        tmpForInput_idx_1 = Vehicle_B.Sum[0];
        tmpForInput_idx_0 = Vehicle_B.Sum[0];
        tmpForInput_idx_2 = tmpForInput_idx_1 * tmpForInput_idx_0;

        /* Product: '<S15>/Divide' */
        Vehicle_B.ORFtoECEF[1] = Vehicle_B.Sum_l[1] / Vehicle_B.MathFunction1;

        /* DotProduct: '<S16>/Dot Product' */
        tmpForInput_idx_1 = Vehicle_B.Sum[1];
        tmpForInput_idx_0 = Vehicle_B.Sum[1];
        tmpForInput_idx_2 += tmpForInput_idx_1 * tmpForInput_idx_0;

        /* Product: '<S15>/Divide' */
        Vehicle_B.ORFtoECEF[2] = Vehicle_B.Sum_l[2] / Vehicle_B.MathFunction1;

        /* DotProduct: '<S16>/Dot Product' */
        tmpForInput_idx_1 = Vehicle_B.Sum[2];
        tmpForInput_idx_0 = Vehicle_B.Sum[2];
        tmpForInput_idx_2 += tmpForInput_idx_1 * tmpForInput_idx_0;

        /* DotProduct: '<S16>/Dot Product' */
        Vehicle_B.DotProduct_p = tmpForInput_idx_2;

        /* Sum: '<S16>/Sum of Elements' */
        tmpForInput_idx_0 = Vehicle_B.DotProduct_p;

        /* Sum: '<S16>/Sum of Elements' */
        Vehicle_B.SumofElements_c = tmpForInput_idx_0;

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

```
              *
              * About '<S16>/Math Function1':
              *  Operator: sqrt
              */
            Vehicle_B.MathFunction1_b = std::sqrt(tmpForInput_idx_0);
          }

          /* End of Math: '<S16>/Math Function1' */

          /* Product: '<S16>/Divide' */
          Vehicle_B.ORFtoECEF[3] = Vehicle_B.Sum[0] / Vehicle_B.MathFunction1_b;

          /* 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' */
          Vehicle_B.ORFtoECEF[4] = Vehicle_B.Sum[1] / Vehicle_B.MathFunction1_b;

          /* 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' */
          Vehicle_B.ORFtoECEF[5] = Vehicle_B.Sum[2] / Vehicle_B.MathFunction1_b;

          /* 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;

          /* DotProduct: '<S17>/Dot Product' */
          Vehicle_B.DotProduct_b = tmpForInput_idx_2;

          /* Sum: '<S17>/Sum of Elements' */
          tmpForInput_idx_0 = Vehicle_B.DotProduct_b;

          /* Sum: '<S17>/Sum of Elements' */
          Vehicle_B.SumofElements_l = tmpForInput_idx_0;

          /* Math: '<S17>/Math Function1'
           *
           * About '<S17>/Math Function1':
           *  Operator: sqrt
           */
          tmpForInput_idx_0 = Vehicle_B.SumofElements_l;
          if (tmpForInput_idx_0 < 0.0) {
            /* Math: '<S17>/Math Function1'
             *
             * About '<S17>/Math Function1':
             *  Operator: sqrt
             */
            Vehicle_B.MathFunction1_m = -std::sqrt(std::abs(tmpForInput_idx_0));
          } else {
            /* Math: '<S17>/Math Function1'
             *
             * About '<S17>/Math Function1':
             *  Operator: sqrt
             */
            Vehicle_B.MathFunction1_m = std::sqrt(tmpForInput_idx_0);
          }

          /* End of Math: '<S17>/Math Function1' */

          /* Product: '<S17>/Divide' */
          Vehicle_B.ORFtoECEF[6] = Vehicle_B.zr[0] / Vehicle_B.MathFunction1_m;
          Vehicle_B.ORFtoECEF[7] = Vehicle_B.zr[1] / Vehicle_B.MathFunction1_m;
          Vehicle_B.ORFtoECEF[8] = Vehicle_B.zr[2] / Vehicle_B.MathFunction1_m;
          for (i = 0; i < 3; i++) {
            /* Math: '<S7>/ECEF to ORF' incorporates:
             *  Concatenate: '<S7>/ORF to ECEF'
             */
            Vehicle_B.ECEFtoORF[3 * i] = Vehicle_B.ORFtoECEF[i];
            Vehicle_B.ECEFtoORF[3 * i + 1] = Vehicle_B.ORFtoECEF[i + 3];
            Vehicle_B.ECEFtoORF[3 * i + 2] = Vehicle_B.ORFtoECEF[i + 6];
          }

          /* Sum: '<S25>/Add' */
          tmpForInput_idx_0 = Vehicle_B.ECEFtoORF[0];
          tmpForInput_idx_1 = Vehicle_B.ECEFtoORF[4];
          tmpForInput_idx_2 = Vehicle_B.ECEFtoORF[8];
          tmpForInput_idx_0 += tmpForInput_idx_1;
          tmpForInput_idx_0 += tmpForInput_idx_2;

          /* Sum: '<S25>/Add' */
          Vehicle_B.Add = tmpForInput_idx_0;

          /* If: '<S14>/If' incorporates:
           *  If: '<S22>/Find Maximum Diagonal Value'
           */
          if (Vehicle_B.Add > 0.0) {
            /* Outputs for IfAction SubSystem: '<S14>/Positive Trace' incorporates:
             *  ActionPort: '<S23>/Action Port'
             */
            /* Sum: '<S23>/Sum' incorporates:
             *  Constant: '<S23>/Constant'
             */
            Vehicle_B.Sum_gh = Vehicle_B.Add + 1.0;

            /* Sqrt: '<S23>/sqrt' */
            Vehicle_B.sqrt_a = std::sqrt(Vehicle_B.Sum_gh);

```

```
        /* Gain: '<S23>/Gain' */
        Vehicle_B.Merge_i[0] = 0.5 * Vehicle_B.sqrt_a;

        /* Gain: '<S23>/Gain1' */
        Vehicle_B.Gain1 = 2.0 * Vehicle_B.sqrt_a;

        /* Sum: '<S46>/Add' */
        Vehicle_B.Add_k = Vehicle_B.ECEFtoORF[7] - Vehicle_B.ECEFtoORF[5];

        /* Sum: '<S45>/Add' */
        Vehicle_B.Add_el = Vehicle_B.ECEFtoORF[2] - Vehicle_B.ECEFtoORF[6];

        /* Sum: '<S47>/Add' */
        Vehicle_B.Add_a = Vehicle_B.ECEFtoORF[3] - Vehicle_B.ECEFtoORF[1];

        /* Product: '<S23>/Product' */
        Vehicle_B.Merge_i[1] = Vehicle_B.Add_k / Vehicle_B.Gain1;
        Vehicle_B.Merge_i[2] = Vehicle_B.Add_el / Vehicle_B.Gain1;
        Vehicle_B.Merge_i[3] = Vehicle_B.Add_a / Vehicle_B.Gain1;

        /* End of Outputs for SubSystem: '<S14>/Positive Trace' */
      } else {
        /* Outputs for IfAction SubSystem: '<S14>/Negative Trace' incorporates:
         *  ActionPort: '<S22>/Action Port'
         */
        if ((Vehicle_B.ECEFtoORF[4] > Vehicle_B.ECEFtoORF[0]) &&
            (Vehicle_B.ECEFtoORF[4] > Vehicle_B.ECEFtoORF[8])) {
          /* Outputs for IfAction SubSystem: '<S22>/Maximum Value at DCM(2,2)' incorporates:
           *  ActionPort: '<S27>/Action Port'
           */
          /* If: '<S22>/Find Maximum Diagonal Value' incorporates:
           *  Constant: '<S39>/Constant'
           *  Gain: '<S27>/Gain'
           *  Gain: '<S27>/Gain1'
           *  Gain: '<S27>/Gain3'
           *  Gain: '<S27>/Gain4'
           *  Product: '<S27>/Product'
           *  Product: '<S38>/Product'
           *  Sqrt: '<S27>/sqrt'
           *  Sum: '<S35>/Add'
           *  Sum: '<S36>/Add'
           *  Sum: '<S37>/Add'
           *  Sum: '<S39>/Add'
           *  Switch: '<S38>/Switch'
           */
          Vehicle_B.Add_g = ((Vehicle_B.ECEFtoORF[4] - Vehicle_B.ECEFtoORF[0]) -
                             Vehicle_B.ECEFtoORF[8]) + 1.0;
          Vehicle_B.sqrt_mw = std::sqrt(Vehicle_B.Add_g);
          Vehicle_B.Merge_i[2] = 0.5 * Vehicle_B.sqrt_mw;
          Vehicle_B.Add_gw = Vehicle_B.ECEFtoORF[1] + Vehicle_B.ECEFtoORF[3];
          Vehicle_B.Add_fc = Vehicle_B.ECEFtoORF[5] + Vehicle_B.ECEFtoORF[7];
          Vehicle_B.Add_d = Vehicle_B.ECEFtoORF[2] - Vehicle_B.ECEFtoORF[6];
          if (Vehicle_B.sqrt_mw != 0.0) {
            /* Switch: '<S38>/Switch' incorporates:
             *  Constant: '<S38>/Constant1'
             */
            Vehicle_B.Switch_f[0] = 0.5;
            Vehicle_B.Switch_f[1] = Vehicle_B.sqrt_mw;
          } else {
            /* Switch: '<S38>/Switch' */
            Vehicle_B.Switch_f[0] = 0.0;
            Vehicle_B.Switch_f[1] = 1.0;
          }

          Vehicle_B.Product_e = Vehicle_B.Switch_f[0] / Vehicle_B.Switch_f[1];
          Vehicle_B.Product_o[0] = Vehicle_B.Add_gw * Vehicle_B.Product_e;
          Vehicle_B.Product_o[1] = Vehicle_B.Add_fc * Vehicle_B.Product_e;
          Vehicle_B.Product_o[2] = Vehicle_B.Add_d * Vehicle_B.Product_e;
          Vehicle_B.Merge_i[1] = Vehicle_B.Product_o[0];
          Vehicle_B.Merge_i[3] = Vehicle_B.Product_o[1];
          Vehicle_B.Merge_i[0] = Vehicle_B.Product_o[2];

          /* End of Outputs for SubSystem: '<S22>/Maximum Value at DCM(2,2)' */
        } else if (Vehicle_B.ECEFtoORF[8] > Vehicle_B.ECEFtoORF[0]) {
          /* Outputs for IfAction SubSystem: '<S22>/Maximum Value at DCM(3,3)' incorporates:
           *  ActionPort: '<S28>/Action Port'
           */
          /* If: '<S22>/Find Maximum Diagonal Value' incorporates:
           *  Constant: '<S44>/Constant'
           *  Gain: '<S28>/Gain'
           *  Gain: '<S28>/Gain1'
           *  Gain: '<S28>/Gain2'
           *  Gain: '<S28>/Gain3'
           *  Product: '<S28>/Product'
           *  Product: '<S43>/Product'
           *  Sqrt: '<S28>/sqrt'
           *  Sum: '<S40>/Add'
           *  Sum: '<S41>/Add'
           *  Sum: '<S42>/Add'
           *  Sum: '<S44>/Add'
           *  Switch: '<S43>/Switch'
           */
          Vehicle_B.Add_f = ((Vehicle_B.ECEFtoORF[8] - Vehicle_B.ECEFtoORF[0]) -
                             Vehicle_B.ECEFtoORF[4]) + 1.0;
          Vehicle_B.sqrt_l = std::sqrt(Vehicle_B.Add_f);
          Vehicle_B.Merge_i[3] = 0.5 * Vehicle_B.sqrt_l;
          Vehicle_B.Add_e = Vehicle_B.ECEFtoORF[2] + Vehicle_B.ECEFtoORF[6];
          Vehicle_B.Add_l2 = Vehicle_B.ECEFtoORF[5] + Vehicle_B.ECEFtoORF[7];
          Vehicle_B.Add_ek = Vehicle_B.ECEFtoORF[3] - Vehicle_B.ECEFtoORF[1];
          if (Vehicle_B.sqrt_l != 0.0) {
            /* Switch: '<S43>/Switch' incorporates:
             *  Constant: '<S43>/Constant1'
             */
            Vehicle_B.Switch_b[0] = 0.5;
```

```
          Vehicle_B.Switch_b[1] = Vehicle_B.sqrt_l;
        } else {
          /* Switch: '<S43>/Switch' */
          Vehicle_B.Switch_b[0] = 0.0;
          Vehicle_B.Switch_b[1] = 1.0;
        }

        Vehicle_B.Product_d = Vehicle_B.Switch_b[0] / Vehicle_B.Switch_b[1];
        Vehicle_B.Product_i[0] = Vehicle_B.Add_e * Vehicle_B.Product_d;
        Vehicle_B.Product_i[1] = Vehicle_B.Add_l2 * Vehicle_B.Product_d;
        Vehicle_B.Product_i[2] = Vehicle_B.Add_ek * Vehicle_B.Product_d;
        Vehicle_B.Merge_i[1] = Vehicle_B.Product_i[0];
        Vehicle_B.Merge_i[2] = Vehicle_B.Product_i[1];
        Vehicle_B.Merge_i[0] = Vehicle_B.Product_i[2];

        /* End of Outputs for SubSystem: '<S22>/Maximum Value at DCM(3,3)' */
      } else {
        /* Outputs for IfAction SubSystem: '<S22>/Maximum Value at DCM(1,1)' incorporates:
         *  ActionPort: '<S26>/Action Port'
         */
        /* If: '<S22>/Find Maximum Diagonal Value' incorporates:
         *  Constant: '<S34>/Constant'
         *  Gain: '<S26>/Gain'
         *  Gain: '<S26>/Gain1'
         *  Gain: '<S26>/Gain2'
         *  Gain: '<S26>/Gain3'
         *  Product: '<S26>/Product'
         *  Product: '<S33>/Product'
         *  Sqrt: '<S26>/sqrt'
         *  Sum: '<S30>/Add'
         *  Sum: '<S31>/Add'
         *  Sum: '<S32>/Add'
         *  Sum: '<S34>/Add'
         *  Switch: '<S33>/Switch'
         */
        Vehicle_B.Add_m = ((Vehicle_B.ECEFtoORF[0] - Vehicle_B.ECEFtoORF[4]) -
                           Vehicle_B.ECEFtoORF[8]) + 1.0;
        Vehicle_B.sqrt_m = std::sqrt(Vehicle_B.Add_m);
        Vehicle_B.Merge_i[1] = 0.5 * Vehicle_B.sqrt_m;
        if (Vehicle_B.sqrt_m != 0.0) {
          /* Switch: '<S33>/Switch' incorporates:
           *  Constant: '<S33>/Constant1'
           */
          Vehicle_B.Switch[0] = 0.5;
          Vehicle_B.Switch[1] = Vehicle_B.sqrt_m;
        } else {
          /* Switch: '<S33>/Switch' */
          Vehicle_B.Switch[0] = 0.0;
          Vehicle_B.Switch[1] = 1.0;
        }

        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_ov = Vehicle_B.ECEFtoORF[2] + Vehicle_B.ECEFtoORF[6];
        Vehicle_B.Add_l = Vehicle_B.ECEFtoORF[7] - Vehicle_B.ECEFtoORF[5];
        Vehicle_B.Product_cy[0] = Vehicle_B.Product_k * Vehicle_B.Add_o;
        Vehicle_B.Product_cy[1] = Vehicle_B.Product_k * Vehicle_B.Add_ov;
        Vehicle_B.Product_cy[2] = Vehicle_B.Product_k * Vehicle_B.Add_l;
        Vehicle_B.Merge_i[2] = Vehicle_B.Product_cy[0];
        Vehicle_B.Merge_i[3] = Vehicle_B.Product_cy[1];
        Vehicle_B.Merge_i[0] = Vehicle_B.Product_cy[2];

        /* End of Outputs for SubSystem: '<S22>/Maximum Value at DCM(1,1)' */
      }

      /* End of Outputs for SubSystem: '<S14>/Negative Trace' */
    }

    /* End of If: '<S14>/If' */

    /* Product: '<S111>/Product' incorporates:
     *  Inport: '<Root>/qECEF2b'
     */
    Vehicle_B.Product = Vehicle_U.qECEF2b[0] * Vehicle_U.qECEF2b[0];

    /* Product: '<S111>/Product1' incorporates:
     *  Inport: '<Root>/qECEF2b'
     */
    Vehicle_B.Product1 = Vehicle_U.qECEF2b[1] * Vehicle_U.qECEF2b[1];

    /* Product: '<S111>/Product2' incorporates:
     *  Inport: '<Root>/qECEF2b'
     */
    Vehicle_B.Product2 = Vehicle_U.qECEF2b[2] * Vehicle_U.qECEF2b[2];

    /* Product: '<S111>/Product3' incorporates:
     *  Inport: '<Root>/qECEF2b'
     */
    Vehicle_B.Product3 = Vehicle_U.qECEF2b[3] * Vehicle_U.qECEF2b[3];

    /* Sum: '<S111>/Sum' */
    Vehicle_B.Sum_i = ((Vehicle_B.Product + Vehicle_B.Product1) +
                       Vehicle_B.Product2) + Vehicle_B.Product3;

    /* Product: '<S9>/Divide' incorporates:
     *  Inport: '<Root>/qECEF2b'
     */
    Vehicle_B.Divide = Vehicle_U.qECEF2b[0] / Vehicle_B.Sum_i;

    /* Product: '<S112>/Product' */
    Vehicle_B.Product_j = Vehicle_B.Divide * Vehicle_B.Merge_i[0];

    /* UnaryMinus: '<S110>/Unary Minus' incorporates:
     *  Inport: '<Root>/qECEF2b'
```

```
     */
    Vehicle_B.UnaryMinus = -Vehicle_U.qECEF2b[1];

    /* Product: '<S9>/Divide1' */
    Vehicle_B.Divide1 = Vehicle_B.UnaryMinus / Vehicle_B.Sum_i;

    /* Product: '<S112>/Product1' */
    Vehicle_B.Product1_h = Vehicle_B.Divide1 * Vehicle_B.Merge_i[1];

    /* UnaryMinus: '<S110>/Unary Minus1' incorporates:
     *  Inport: '<Root>/qECEF2b'
     */
    Vehicle_B.UnaryMinus1 = -Vehicle_U.qECEF2b[2];

    /* Product: '<S9>/Divide2' */
    Vehicle_B.Divide2 = Vehicle_B.UnaryMinus1 / Vehicle_B.Sum_i;

    /* Product: '<S112>/Product2' */
    Vehicle_B.Product2_d = Vehicle_B.Divide2 * Vehicle_B.Merge_i[2];

    /* UnaryMinus: '<S110>/Unary Minus2' incorporates:
     *  Inport: '<Root>/qECEF2b'
     */
    Vehicle_B.UnaryMinus2 = -Vehicle_U.qECEF2b[3];

    /* Product: '<S9>/Divide3' */
    Vehicle_B.Divide3 = Vehicle_B.UnaryMinus2 / Vehicle_B.Sum_i;

    /* Product: '<S112>/Product3' */
    Vehicle_B.Product3_c = Vehicle_B.Divide3 * Vehicle_B.Merge_i[3];

    /* Sum: '<S112>/Sum' */
    Vehicle_B.Sum_p = ((Vehicle_B.Product_j - Vehicle_B.Product1_h) -
                       Vehicle_B.Product2_d) - Vehicle_B.Product3_c;

    /* Product: '<S123>/Product' */
    Vehicle_B.Product_b = Vehicle_B.Sum_p * Vehicle_B.Sum_p;

    /* Product: '<S113>/Product' */
    Vehicle_B.Product_p = Vehicle_B.Divide * Vehicle_B.Merge_i[1];

    /* Product: '<S113>/Product1' */
    Vehicle_B.Product1_i = Vehicle_B.Divide1 * Vehicle_B.Merge_i[0];

    /* Product: '<S113>/Product2' */
    Vehicle_B.Product2_o = Vehicle_B.Divide2 * Vehicle_B.Merge_i[3];

    /* Product: '<S113>/Product3' */
    Vehicle_B.Product3_p = Vehicle_B.Divide3 * Vehicle_B.Merge_i[2];

    /* Sum: '<S113>/Sum' */
    Vehicle_B.Sum_iw = ((Vehicle_B.Product_p + Vehicle_B.Product1_i) +
                        Vehicle_B.Product2_o) - Vehicle_B.Product3_p;

    /* Product: '<S123>/Product1' */
    Vehicle_B.Product1_in = Vehicle_B.Sum_iw * Vehicle_B.Sum_iw;

    /* Product: '<S114>/Product' */
    Vehicle_B.Product_by = Vehicle_B.Divide * Vehicle_B.Merge_i[2];

    /* Product: '<S114>/Product1' */
    Vehicle_B.Product1_f = Vehicle_B.Divide1 * Vehicle_B.Merge_i[3];

    /* Product: '<S114>/Product2' */
    Vehicle_B.Product2_m = Vehicle_B.Divide2 * Vehicle_B.Merge_i[0];

    /* Product: '<S114>/Product3' */
    Vehicle_B.Product3_pa = Vehicle_B.Divide3 * Vehicle_B.Merge_i[1];

    /* Sum: '<S114>/Sum' */
    Vehicle_B.Sum_d = ((Vehicle_B.Product_by - Vehicle_B.Product1_f) +
                       Vehicle_B.Product2_m) + Vehicle_B.Product3_pa;

    /* Product: '<S123>/Product2' */
    Vehicle_B.Product2_n = Vehicle_B.Sum_d * Vehicle_B.Sum_d;

    /* Product: '<S115>/Product' */
    Vehicle_B.Product_pd = Vehicle_B.Divide * Vehicle_B.Merge_i[3];

    /* Product: '<S115>/Product1' */
    Vehicle_B.Product1_d = Vehicle_B.Divide1 * Vehicle_B.Merge_i[2];

    /* Product: '<S115>/Product2' */
    Vehicle_B.Product2_nn = Vehicle_B.Divide2 * Vehicle_B.Merge_i[1];

    /* Product: '<S115>/Product3' */
    Vehicle_B.Product3_l = Vehicle_B.Divide3 * Vehicle_B.Merge_i[0];

    /* Sum: '<S115>/Sum' */
    Vehicle_B.Sum_o = ((Vehicle_B.Product_pd + Vehicle_B.Product1_d) -
                       Vehicle_B.Product2_nn) + Vehicle_B.Product3_l;

    /* Product: '<S123>/Product3' */
    Vehicle_B.Product3_ph = Vehicle_B.Sum_o * Vehicle_B.Sum_o;

    /* Sum: '<S123>/Sum' */
    Vehicle_B.Sum_g = ((Vehicle_B.Product_b + Vehicle_B.Product1_in) +
                       Vehicle_B.Product2_n) + Vehicle_B.Product3_ph;

    /* Sqrt: '<S122>/sqrt' */
    Vehicle_B.sqrt_g = std::sqrt(Vehicle_B.Sum_g);

    /* Product: '<S117>/Product' */
    Vehicle_B.Product_h = Vehicle_B.Sum_p / Vehicle_B.sqrt_g;
```

```
    /* Product: '<S117>/Product1' */
    Vehicle_B.Product1_j = Vehicle_B.Sum_iw / Vehicle_B.sqrt_g;

    /* Product: '<S117>/Product2' */
    Vehicle_B.Product2_dj = Vehicle_B.Sum_d / Vehicle_B.sqrt_g;

    /* Product: '<S117>/Product3' */
    Vehicle_B.Product3_e = Vehicle_B.Sum_o / Vehicle_B.sqrt_g;

    /* Fcn: '<S11>/fcn1' */
    Vehicle_B.fcn1 = (Vehicle_B.Product2_dj * Vehicle_B.Product3_e -
                      Vehicle_B.Product_h * Vehicle_B.Product1_j) * -2.0;

    /* Fcn: '<S11>/fcn2' */
    Vehicle_B.fcn2 = ((Vehicle_B.Product_h * Vehicle_B.Product_h -
                       Vehicle_B.Product1_j * Vehicle_B.Product1_j) -
                      Vehicle_B.Product2_dj * Vehicle_B.Product2_dj) +
      Vehicle_B.Product3_e * Vehicle_B.Product3_e;

    /* Trigonometry: '<S116>/Trigonometric Function1' */
    Vehicle_B.VectorConcatenate_a[0] = rt_atan2d_snf(Vehicle_B.fcn1,
      Vehicle_B.fcn2);

    /* Fcn: '<S11>/fcn3' */
    Vehicle_B.fcn3 = (Vehicle_B.Product1_j * Vehicle_B.Product3_e +
                      Vehicle_B.Product_h * Vehicle_B.Product2_dj) * 2.0;

    /* If: '<S118>/If' */
    if (Vehicle_B.fcn3 > 1.0) {
      /* Outputs for IfAction SubSystem: '<S118>/If Action Subsystem' incorporates:
       *  ActionPort: '<S119>/Action Port'
       */
      /* Merge: '<S118>/Merge' incorporates:
       *  Constant: '<S119>/Constant'
       */
      Vehicle_B.Merge_il = 1.0;

      /* End of Outputs for SubSystem: '<S118>/If Action Subsystem' */
    } else if (Vehicle_B.fcn3 < -1.0) {
      /* Outputs for IfAction SubSystem: '<S118>/If Action Subsystem1' incorporates:
       *  ActionPort: '<S120>/Action Port'
       */
      /* Merge: '<S118>/Merge' incorporates:
       *  Constant: '<S120>/Constant'
       */
      Vehicle_B.Merge_il = 1.0;

      /* End of Outputs for SubSystem: '<S118>/If Action Subsystem1' */
    } else {
      /* Outputs for IfAction SubSystem: '<S118>/If Action Subsystem2' incorporates:
       *  ActionPort: '<S121>/Action Port'
       */
      //Vehicle_IfActionSubsystem2_f(Vehicle_B.fcn3, &Vehicle_B.Merge_il);
      Vehicle_B.Merge_il = Vehicle_B.fcn3;
      /* End of Outputs for SubSystem: '<S118>/If Action Subsystem2' */
    }

    /* End of If: '<S118>/If' */

    /* Trigonometry: '<S116>/trigFcn' */
    tmpForInput_idx_0 = Vehicle_B.Merge_il;
    if (tmpForInput_idx_0 > 1.0) {
      tmpForInput_idx_0 = 1.0;
    } else {
      if (tmpForInput_idx_0 < -1.0) {
        tmpForInput_idx_0 = -1.0;
      }
    }

    Vehicle_B.VectorConcatenate_a[1] = std::asin(tmpForInput_idx_0);

    /* End of Trigonometry: '<S116>/trigFcn' */

    /* Fcn: '<S11>/fcn4' */
    Vehicle_B.fcn4 = (Vehicle_B.Product1_j * Vehicle_B.Product2_dj -
                      Vehicle_B.Product_h * Vehicle_B.Product3_e) * -2.0;

    /* Fcn: '<S11>/fcn5' */
    Vehicle_B.fcn5 = ((Vehicle_B.Product_h * Vehicle_B.Product_h +
                       Vehicle_B.Product1_j * Vehicle_B.Product1_j) -
                      Vehicle_B.Product2_dj * Vehicle_B.Product2_dj) -
      Vehicle_B.Product3_e * Vehicle_B.Product3_e;

    /* Trigonometry: '<S116>/Trigonometric Function3' */
    Vehicle_B.VectorConcatenate_a[2] = rt_atan2d_snf(Vehicle_B.fcn4,
      Vehicle_B.fcn5);

    /* Gain: '<S85>/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 * 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[0] = Vehicle_B.DerivativeGain[0];

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

```
    Vehicle_B.UD[0] = Vehicle_DW.UD_DSTATE[0];

    /* Sum: '<S86>/Diff' */
    Vehicle_B.Diff[0] = Vehicle_B.Tsamp[0] - Vehicle_B.UD[0];

    /* Gain: '<S90>/Integral Gain' */
    Vehicle_B.IntegralGain[0] = 2.0E-10 * Vehicle_B.VectorConcatenate_a[0];

    /* DiscreteIntegrator: '<S93>/Integrator' */
    Vehicle_B.Integrator[0] = Vehicle_DW.Integrator_DSTATE[0];

    /* Gain: '<S98>/Proportional Gain' */
    Vehicle_B.ProportionalGain[0] = 10.0E-5 * Vehicle_B.VectorConcatenate_a[0];

    /* Merge: '<S1>/Merge' incorporates:
     *  Sum: '<S102>/Sum'
     */
    Vehicle_B.Merge[0] = (Vehicle_B.ProportionalGain[0] + Vehicle_B.Integrator[0])
      + Vehicle_B.Diff[0];

    /* Update for Delay: '<S86>/UD' */
    Vehicle_DW.UD_DSTATE[0] = Vehicle_B.Tsamp[0];

    /* Update for DiscreteIntegrator: '<S93>/Integrator' */
    Vehicle_DW.Integrator_DSTATE[0] += Vehicle_B.IntegralGain[0];

    /* Gain: '<S85>/Derivative Gain' */
    Vehicle_B.DerivativeGain[1] = 0.02 * Vehicle_B.VectorConcatenate_a[1];

    /* SampleTimeMath: '<S88>/Tsamp'
     *
     * About '<S88>/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[1] = Vehicle_B.DerivativeGain[1];

    /* Delay: '<S86>/UD' */
    Vehicle_B.UD[1] = Vehicle_DW.UD_DSTATE[1];

    /* Sum: '<S86>/Diff' */
    Vehicle_B.Diff[1] = Vehicle_B.Tsamp[1] - Vehicle_B.UD[1];

    /* Gain: '<S90>/Integral Gain' */
    Vehicle_B.IntegralGain[1] = 2.0E-10 * Vehicle_B.VectorConcatenate_a[1];

    /* DiscreteIntegrator: '<S93>/Integrator' */
    Vehicle_B.Integrator[1] = Vehicle_DW.Integrator_DSTATE[1];

    /* Gain: '<S98>/Proportional Gain' */
    Vehicle_B.ProportionalGain[1] = 10.0E-5 * Vehicle_B.VectorConcatenate_a[1];

    /* Merge: '<S1>/Merge' incorporates:
     *  Sum: '<S102>/Sum'
     */
    Vehicle_B.Merge[1] = (Vehicle_B.ProportionalGain[1] + Vehicle_B.Integrator[1])
      + Vehicle_B.Diff[1];

    /* Update for Delay: '<S86>/UD' */
    Vehicle_DW.UD_DSTATE[1] = Vehicle_B.Tsamp[1];

    /* Update for DiscreteIntegrator: '<S93>/Integrator' */
    Vehicle_DW.Integrator_DSTATE[1] += Vehicle_B.IntegralGain[1];

    /* Gain: '<S85>/Derivative Gain' */
    Vehicle_B.DerivativeGain[2] = 0.02 * Vehicle_B.VectorConcatenate_a[2];

    /* SampleTimeMath: '<S88>/Tsamp'
     *
     * About '<S88>/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[2] = Vehicle_B.DerivativeGain[2];

    /* Delay: '<S86>/UD' */
    Vehicle_B.UD[2] = Vehicle_DW.UD_DSTATE[2];

    /* Sum: '<S86>/Diff' */
    Vehicle_B.Diff[2] = Vehicle_B.Tsamp[2] - Vehicle_B.UD[2];

    /* Gain: '<S90>/Integral Gain' */
    Vehicle_B.IntegralGain[2] = 2.0E-10 * Vehicle_B.VectorConcatenate_a[2];

    /* DiscreteIntegrator: '<S93>/Integrator' */
    Vehicle_B.Integrator[2] = Vehicle_DW.Integrator_DSTATE[2];

    /* Gain: '<S98>/Proportional Gain' */
    Vehicle_B.ProportionalGain[2] = 10.0E-5 * Vehicle_B.VectorConcatenate_a[2];

    /* Merge: '<S1>/Merge' incorporates:
     *  Sum: '<S102>/Sum'
     */
    Vehicle_B.Merge[2] = (Vehicle_B.ProportionalGain[2] + Vehicle_B.Integrator[2])
      + Vehicle_B.Diff[2];

```

```
        /* Update for Delay: '<S86>/UD' */
        Vehicle_DW.UD_DSTATE[2] = Vehicle_B.Tsamp[2];

        /* Update for DiscreteIntegrator: '<S93>/Integrator' */
        Vehicle_DW.Integrator_DSTATE[2] += Vehicle_B.IntegralGain[2];

        /* End of Outputs for SubSystem: '<S1>/NadirPointing Mode' */
}
```

SunPointing.cpp

```
#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;
        /* Outputs for IfAction SubSystem: '<S1>/SunPointing Mode' incorporates:
         *  ActionPort: '<S4>/Action Port'
         */
        /* If: '<S129>/If' */
        if (Vehicle_ConstB.Abs < 1.0E-6) {
          /* Outputs for IfAction SubSystem: '<S129>/If Action Subsystem' incorporates:
           *  ActionPort: '<S140>/Action Port'
           */
          //Vehicle_IfActionSubsystem(&Vehicle_B.Merge_m);
            Vehicle_B.Merge_m = 1.0;
          /* End of Outputs for SubSystem: '<S129>/If Action Subsystem' */
        } else if (Vehicle_ConstB.Abs1 < 1.0E-6) {
          /* Outputs for IfAction SubSystem: '<S129>/If Action Subsystem1' incorporates:
           *  ActionPort: '<S141>/Action Port'
           */
         // Vehicle_IfActionSubsystem1(&Vehicle_B.Merge_m);
            Vehicle_B.Merge_m = -1.0;
          /* End of Outputs for SubSystem: '<S129>/If Action Subsystem1' */
        } else {
          /* Outputs for IfAction SubSystem: '<S129>/If Action Subsystem2' incorporates:
           *  ActionPort: '<S142>/Action Port'
           */
          //Vehicle_IfActionSubsystem2(&Vehicle_B.Merge_m);
            Vehicle_B.Merge_m = 0.0;
          /* End of Outputs for SubSystem: '<S129>/If Action Subsystem2' */
        }

        /* End of If: '<S129>/If' */

        /* Product: '<S259>/Product' incorporates:
         *  Inport: '<Root>/q_eci2b'
         */
        Vehicle_B.Product_n = Vehicle_U.q_eci2b[0] * Vehicle_U.q_eci2b[0];

        /* Product: '<S259>/Product1' incorporates:
         *  Inport: '<Root>/q_eci2b'
         */
        Vehicle_B.Product1_ii = Vehicle_U.q_eci2b[1] * Vehicle_U.q_eci2b[1];

        /* Product: '<S259>/Product2' incorporates:
         *  Inport: '<Root>/q_eci2b'
         */
        Vehicle_B.Product2_c = Vehicle_U.q_eci2b[2] * Vehicle_U.q_eci2b[2];

        /* Product: '<S259>/Product3' incorporates:
         *  Inport: '<Root>/q_eci2b'
         */
        Vehicle_B.Product3_d = Vehicle_U.q_eci2b[3] * Vehicle_U.q_eci2b[3];

        /* Sum: '<S259>/Sum' */
        Vehicle_B.Sum_j = ((Vehicle_B.Product_n + Vehicle_B.Product1_ii) +
                           Vehicle_B.Product2_c) + Vehicle_B.Product3_d;

        /* Sqrt: '<S258>/sqrt' */
        Vehicle_B.sqrt_n = std::sqrt(Vehicle_B.Sum_j);

        /* Product: '<S254>/Product2' incorporates:
         *  Inport: '<Root>/q_eci2b'
         */
        Vehicle_B.Product2_oi = Vehicle_U.q_eci2b[2] / Vehicle_B.sqrt_n;

        /* Product: '<S255>/Product6' */
        Vehicle_B.Product6 = Vehicle_B.Product2_oi * Vehicle_B.Product2_oi;

        /* Product: '<S254>/Product3' incorporates:
         *  Inport: '<Root>/q_eci2b'
         */
        Vehicle_B.Product3_a = Vehicle_U.q_eci2b[3] / Vehicle_B.sqrt_n;

        /* Product: '<S255>/Product7' */
        Vehicle_B.Product7 = Vehicle_B.Product3_a * Vehicle_B.Product3_a;

        /* Sum: '<S255>/Sum3' incorporates:
         *  Constant: '<S255>/Constant'
         */
        Vehicle_B.Sum3 = (0.5 - Vehicle_B.Product6) - Vehicle_B.Product7;

        /* Gain: '<S255>/Gain2' */
        Vehicle_B.Gain2 = 2.0 * Vehicle_B.Sum3;

        /* Product: '<S255>/Product8' incorporates:
```

```
   *  Inport: '<Root>/xSunECI'
   */
  Vehicle_B.Product8 = Vehicle_U.xSunECI[0] * Vehicle_B.Gain2;

  /* Product: '<S254>/Product1' incorporates:
   *  Inport: '<Root>/q_eci2b'
   */
  Vehicle_B.Product1_fv = Vehicle_U.q_eci2b[1] / Vehicle_B.sqrt_n;

  /* Product: '<S255>/Product' */
  Vehicle_B.Product_jb = Vehicle_B.Product1_fv * Vehicle_B.Product2_oi;

  /* Product: '<S254>/Product' incorporates:
   *  Inport: '<Root>/q_eci2b'
   */
  Vehicle_B.Product_cw = Vehicle_U.q_eci2b[0] / Vehicle_B.sqrt_n;

  /* Product: '<S255>/Product1' */
  Vehicle_B.Product1_b = Vehicle_B.Product_cw * Vehicle_B.Product3_a;

  /* Sum: '<S255>/Sum1' */
  Vehicle_B.Sum1 = Vehicle_B.Product_jb + Vehicle_B.Product1_b;

  /* Gain: '<S255>/Gain' */
  Vehicle_B.Gain = 2.0 * Vehicle_B.Sum1;

  /* Product: '<S255>/Product4' incorporates:
   *  Inport: '<Root>/xSunECI'
   */
  Vehicle_B.Product4_p = Vehicle_B.Gain * Vehicle_U.xSunECI[1];

  /* Product: '<S255>/Product2' */
  Vehicle_B.Product2_ox = Vehicle_B.Product_cw * Vehicle_B.Product2_oi;

  /* Product: '<S255>/Product3' */
  Vehicle_B.Product3_pv = Vehicle_B.Product1_fv * Vehicle_B.Product3_a;

  /* Sum: '<S255>/Sum2' */
  Vehicle_B.Sum2 = Vehicle_B.Product3_pv - Vehicle_B.Product2_ox;

  /* Gain: '<S255>/Gain1' */
  Vehicle_B.Gain1_l = 2.0 * Vehicle_B.Sum2;

  /* Product: '<S255>/Product5' incorporates:
   *  Inport: '<Root>/xSunECI'
   */
  Vehicle_B.Product5_g = Vehicle_B.Gain1_l * Vehicle_U.xSunECI[2];

  /* Sum: '<S255>/Sum' */
  Vehicle_B.Sum_k = (Vehicle_B.Product8 + Vehicle_B.Product4_p) +
    Vehicle_B.Product5_g;

  /* Product: '<S256>/Product' */
  Vehicle_B.Product_if = Vehicle_B.Product1_fv * Vehicle_B.Product2_oi;

  /* Product: '<S256>/Product1' */
  Vehicle_B.Product1_e = Vehicle_B.Product_cw * Vehicle_B.Product3_a;

  /* Sum: '<S256>/Sum1' */
  Vehicle_B.Sum1_b = Vehicle_B.Product_if - Vehicle_B.Product1_e;

  /* Gain: '<S256>/Gain' */
  Vehicle_B.Gain_a = 2.0 * Vehicle_B.Sum1_b;

  /* Product: '<S256>/Product4' incorporates:
   *  Inport: '<Root>/xSunECI'
   */
  Vehicle_B.Product4_k = Vehicle_U.xSunECI[0] * Vehicle_B.Gain_a;

  /* Product: '<S256>/Product6' */
  Vehicle_B.Product6_d = Vehicle_B.Product1_fv * Vehicle_B.Product1_fv;

  /* Product: '<S256>/Product7' */
  Vehicle_B.Product7_e = Vehicle_B.Product3_a * Vehicle_B.Product3_a;

  /* Sum: '<S256>/Sum3' incorporates:
   *  Constant: '<S256>/Constant'
   */
  Vehicle_B.Sum3_p = (0.5 - Vehicle_B.Product6_d) - Vehicle_B.Product7_e;

  /* Gain: '<S256>/Gain2' */
  Vehicle_B.Gain2_a = 2.0 * Vehicle_B.Sum3_p;

  /* Product: '<S256>/Product8' incorporates:
   *  Inport: '<Root>/xSunECI'
   */
  Vehicle_B.Product8_c = Vehicle_B.Gain2_a * Vehicle_U.xSunECI[1];

  /* Product: '<S256>/Product2' */
  Vehicle_B.Product2_l = Vehicle_B.Product_cw * Vehicle_B.Product1_fv;

  /* Product: '<S256>/Product3' */
  Vehicle_B.Product3_c5 = Vehicle_B.Product2_oi * Vehicle_B.Product3_a;

  /* Sum: '<S256>/Sum2' */
  Vehicle_B.Sum2_k = Vehicle_B.Product2_l + Vehicle_B.Product3_c5;

  /* Gain: '<S256>/Gain1' */
  Vehicle_B.Gain1_d = 2.0 * Vehicle_B.Sum2_k;

  /* Product: '<S256>/Product5' incorporates:
   *  Inport: '<Root>/xSunECI'
   */
  Vehicle_B.Product5_j = Vehicle_B.Gain1_d * Vehicle_U.xSunECI[2];
```

```
      /* Sum: '<S256>/Sum' */
      Vehicle_B.Sum_lm = (Vehicle_B.Product4_k + Vehicle_B.Product8_c) +
        Vehicle_B.Product5_j;

      /* 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.Sum1_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:
       *  Inport: '<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.Sum2_p = Vehicle_B.Product3_f - Vehicle_B.Product2_k;

      /* Gain: '<S257>/Gain1' */
      Vehicle_B.Gain1_i = 2.0 * Vehicle_B.Sum2_p;

      /* Product: '<S257>/Product5' incorporates:
       *  Inport: '<Root>/xSunECI'
       */
      Vehicle_B.Product5_d = Vehicle_B.Gain1_i * 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:
       *  Inport: '<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_k;
      Vehicle_B.TmpSignalConversionAtDotProduct[1] = Vehicle_B.Sum_lm;
      Vehicle_B.TmpSignalConversionAtDotProduct[2] = Vehicle_B.Sum_n;

      /* Product: '<S265>/Product' incorporates:
       *  Inport: '<Root>/q_eci2b'
       */
      Vehicle_B.Product_jz = Vehicle_U.q_eci2b[0] * Vehicle_U.q_eci2b[0];

      /* Product: '<S265>/Product1' incorporates:
       *  Inport: '<Root>/q_eci2b'
       */
      Vehicle_B.Product1_bp = Vehicle_U.q_eci2b[1] * Vehicle_U.q_eci2b[1];

      /* Product: '<S265>/Product2' incorporates:
       *  Inport: '<Root>/q_eci2b'
       */
      Vehicle_B.Product2_b = Vehicle_U.q_eci2b[2] * Vehicle_U.q_eci2b[2];

      /* Product: '<S265>/Product3' incorporates:
       *  Inport: '<Root>/q_eci2b'
       */
      Vehicle_B.Product3_dh = Vehicle_U.q_eci2b[3] * Vehicle_U.q_eci2b[3];

      /* 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:
       *  Inport: '<Root>/q_eci2b'
       */
      Vehicle_B.Product2_lq = Vehicle_U.q_eci2b[2] / Vehicle_B.sqrt_k;

      /* Product: '<S261>/Product6' */
      Vehicle_B.Product6_l = Vehicle_B.Product2_lq * Vehicle_B.Product2_lq;

      /* Product: '<S260>/Product3' incorporates:
       *  Inport: '<Root>/q_eci2b'
```

```
     */
    Vehicle_B.Product3_h = Vehicle_U.q_eci2b[3] / Vehicle_B.sqrt_k;

    /* 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:
     *  Inport: '<Root>/q_eci2b'
     */
    Vehicle_B.Product1_b1 = Vehicle_U.q_eci2b[1] / Vehicle_B.sqrt_k;

    /* Product: '<S261>/Product' */
    Vehicle_B.Product_h1 = Vehicle_B.Product1_b1 * Vehicle_B.Product2_lq;

    /* Product: '<S260>/Product' incorporates:
     *  Inport: '<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.Product_h1 + 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;
```

```
        /* Product: '<S262>/Product5' */
        Vehicle_B.Product5_b = Vehicle_B.Gain1_oi;

        /* Sum: '<S262>/Sum' */
        Vehicle_B.Sum_ln = (Vehicle_B.Product4_pt + Vehicle_B.Product8_e) +
          Vehicle_B.Product5_b;

        /* Product: '<S263>/Product' */
        Vehicle_B.Product_l = Vehicle_B.Product1_b1 * Vehicle_B.Product3_h;

        /* Product: '<S263>/Product1' */
        Vehicle_B.Product1_id = Vehicle_B.Product_f * Vehicle_B.Product2_lq;

        /* Sum: '<S263>/Sum1' */
        Vehicle_B.Sum1_d1 = Vehicle_B.Product_l + Vehicle_B.Product1_id;

        /* Gain: '<S263>/Gain' */
        Vehicle_B.Gain_o = 2.0 * Vehicle_B.Sum1_d1;

        /* Product: '<S263>/Product4' */
        Vehicle_B.Product4_d = 0.0 * Vehicle_B.Gain_o;

        /* Product: '<S263>/Product2' */
        Vehicle_B.Product2_f = Vehicle_B.Product_f * Vehicle_B.Product1_b1;

        /* Product: '<S263>/Product3' */
        Vehicle_B.Product3_gx = Vehicle_B.Product2_lq * Vehicle_B.Product3_h;

        /* Sum: '<S263>/Sum2' */
        Vehicle_B.Sum2_l = Vehicle_B.Product3_gx - Vehicle_B.Product2_f;

        /* Gain: '<S263>/Gain1' */
        Vehicle_B.Gain1_g = 2.0 * Vehicle_B.Sum2_l;

        /* Product: '<S263>/Product5' */
        Vehicle_B.Product5_n = Vehicle_B.Gain1_g * 0.0;

        /* Product: '<S263>/Product6' */
        Vehicle_B.Product6_j = Vehicle_B.Product1_b1 * Vehicle_B.Product1_b1;

        /* Product: '<S263>/Product7' */
        Vehicle_B.Product7_g = Vehicle_B.Product2_lq * Vehicle_B.Product2_lq;

        /* Sum: '<S263>/Sum3' incorporates:
         *  Constant: '<S263>/Constant'
         */
        Vehicle_B.Sum3_b = (0.5 - Vehicle_B.Product6_j) - Vehicle_B.Product7_g;

        /* Gain: '<S263>/Gain2' */
        Vehicle_B.Gain2_o = 2.0 * Vehicle_B.Sum3_b;

        /* Product: '<S263>/Product8' */
        Vehicle_B.Product8_a = Vehicle_B.Gain2_o;

        /* Sum: '<S263>/Sum' */
        Vehicle_B.Sum_kb = (Vehicle_B.Product4_d + Vehicle_B.Product5_n) +
          Vehicle_B.Product8_a;

        /* SignalConversion generated from: '<S130>/Dot Product2' */
        Vehicle_B.TmpSignalConversionAtDotProdu_d[0] = Vehicle_B.Sum_kk;
        Vehicle_B.TmpSignalConversionAtDotProdu_d[1] = Vehicle_B.Sum_ln;
        Vehicle_B.TmpSignalConversionAtDotProdu_d[2] = Vehicle_B.Sum_kb;

        /* DotProduct: '<S130>/Dot Product3' */
        tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProduct[0];
        tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_d[0];
        tmpForInput_idx_2 = tmpForInput_idx_1 * tmpForInput_idx_0;

        /* DotProduct: '<S130>/Dot Product1' */
        tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProduct[0];
        tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduct[0];
        tmpForInput = tmpForInput_idx_1 * tmpForInput_idx_0;

        /* DotProduct: '<S130>/Dot Product2' */
        tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProdu_d[0];
        tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_d[0];
        tmpForInput_0 = tmpForInput_idx_1 * tmpForInput_idx_0;

        /* DotProduct: '<S130>/Dot Product3' */
        tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProduct[1];
        tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_d[1];
        tmpForInput_idx_2 += tmpForInput_idx_1 * tmpForInput_idx_0;

        /* DotProduct: '<S130>/Dot Product1' */
        tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProduct[1];
        tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduct[1];
        tmpForInput += tmpForInput_idx_1 * tmpForInput_idx_0;

        /* DotProduct: '<S130>/Dot Product2' */
        tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProdu_d[1];
        tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_d[1];
        tmpForInput_0 += tmpForInput_idx_1 * tmpForInput_idx_0;

        /* DotProduct: '<S130>/Dot Product3' */
        tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProduct[2];
        tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_d[2];
        tmpForInput_idx_2 += tmpForInput_idx_1 * tmpForInput_idx_0;

        /* DotProduct: '<S130>/Dot Product1' */
        tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProduct[2];
        tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduct[2];
        tmpForInput += tmpForInput_idx_1 * tmpForInput_idx_0;
```

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

    /* DotProduct: '<S130>/Dot Product3' */
    Vehicle_B.DotProduct3 = tmpForInput_idx_2;

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

    /* DotProduct: '<S130>/Dot Product2' */
    Vehicle_B.DotProduct2 = tmpForInput_0;

    /* Product: '<S130>/Divide1' */
    Vehicle_B.Divide1_k = Vehicle_B.DotProduct1 * Vehicle_B.DotProduct2;

    /* Sqrt: '<S130>/Sqrt3' */
    Vehicle_B.Sqrt3 = std::sqrt(Vehicle_B.Divide1_k);

    /* 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 IfAction SubSystem: '<S130>/If Action Subsystem' incorporates:
       *  ActionPort: '<S143>/Action Port'
       */
     // Vehicle_IfActionSubsystem(&Vehicle_B.Merge_ip);
        Vehicle_B.Merge_ip = 1.0;
      /* End of Outputs for SubSystem: '<S130>/If Action Subsystem' */
    } else if (Vehicle_B.Abs1_d < 1.0E-6) {
      /* Outputs for IfAction SubSystem: '<S130>/If Action Subsystem1' incorporates:
       *  ActionPort: '<S144>/Action Port'
       */
      //Vehicle_IfActionSubsystem1(&Vehicle_B.Merge_ip);
        Vehicle_B.Merge_ip = -1.0;
      /* End of Outputs for SubSystem: '<S130>/If Action Subsystem1' */
    } else {
      /* Outputs for IfAction SubSystem: '<S130>/If Action Subsystem2' incorporates:
       *  ActionPort: '<S145>/Action Port'
       */
     // Vehicle_IfActionSubsystem2(&Vehicle_B.Merge_ip);
        Vehicle_B.Merge_ip = 0.0;
      /* End of Outputs for SubSystem: '<S130>/If Action Subsystem2' */
    }

    /* End of If: '<S130>/If' */

    /* RelationalOperator: '<S131>/Compare' incorporates:
     *  Constant: '<S131>/Constant'
     */
    Vehicle_B.Compare_k = (Vehicle_B.Merge_m != 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_0 = Vehicle_B.TmpSignalConversionAtDotProduct[0];
    tmpForInput = tmpForInput_idx_2 * tmpForInput_idx_0;

    /* DotProduct: '<S163>/Dot Product3' */
    tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduct[0];
    tmpForInput_0 = 0.0 * tmpForInput_idx_0;

    /* DotProduct: '<S163>/Dot Product2' */
    tmpForInput_idx_2 = Vehicle_B.TmpSignalConversionAtDotProduct[0];
    tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduct[0];
    tmpForInput_1 = tmpForInput_idx_2 * tmpForInput_idx_0;

    /* DotProduct: '<S136>/Dot Product3' */
    tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduct[1];
    tmpForInput_idx_1 += 0.0 * tmpForInput_idx_0;

    /* DotProduct: '<S136>/Dot Product2' */
    tmpForInput_idx_2 = Vehicle_B.TmpSignalConversionAtDotProduct[1];
    tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduct[1];
    tmpForInput += tmpForInput_idx_2 * tmpForInput_idx_0;

    /* DotProduct: '<S163>/Dot Product3' */
    tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduct[1];
    tmpForInput_0 += 0.0 * tmpForInput_idx_0;
```

```
      /* DotProduct: '<S163>/Dot Product2' */
      tmpForInput_idx_2 = Vehicle_B.TmpSignalConversionAtDotProduct[1];
      tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduct[1];
      tmpForInput_1 += tmpForInput_idx_2 * tmpForInput_idx_0;

      /* DotProduct: '<S136>/Dot Product3' */
      tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduct[2];
      tmpForInput_idx_1 += -tmpForInput_idx_0;

      /* DotProduct: '<S136>/Dot Product2' */
      tmpForInput_idx_2 = Vehicle_B.TmpSignalConversionAtDotProduct[2];
      tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduct[2];
      tmpForInput += tmpForInput_idx_2 * tmpForInput_idx_0;

      /* DotProduct: '<S163>/Dot Product3' */
      tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduct[2];
      tmpForInput_0 += -tmpForInput_idx_0;

      /* DotProduct: '<S163>/Dot Product2' */
      tmpForInput_idx_2 = Vehicle_B.TmpSignalConversionAtDotProduct[2];
      tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProduct[2];
      tmpForInput_1 += tmpForInput_idx_2 * tmpForInput_idx_0;

      /* DotProduct: '<S136>/Dot Product3' */
      Vehicle_B.DotProduct3_o = tmpForInput_idx_1;

      /* DotProduct: '<S136>/Dot Product2' */
      Vehicle_B.DotProduct2_c = tmpForInput;

      /* Product: '<S136>/Divide1' */
      Vehicle_B.Divide1_d = Vehicle_ConstB.DotProduct1_c * Vehicle_B.DotProduct2_c;

      /* Sqrt: '<S136>/Sqrt3' */
      Vehicle_B.Sqrt3_j = std::sqrt(Vehicle_B.Divide1_d);

      /* Sum: '<S136>/Add2' */
      Vehicle_B.Add2 = Vehicle_B.DotProduct3_o + Vehicle_B.Sqrt3_j;

      /* Product: '<S176>/Product' */
      Vehicle_B.Product_i2 = Vehicle_B.Add2 * Vehicle_B.Add2;

      /* DotProduct: '<S163>/Dot Product3' */
      Vehicle_B.DotProduct3_p = tmpForInput_0;

      /* DotProduct: '<S163>/Dot Product2' */
      Vehicle_B.DotProduct2_f = tmpForInput_1;

      /* Product: '<S163>/Divide1' */
      Vehicle_B.Divide1_b = Vehicle_ConstB.DotProduct1_g * Vehicle_B.DotProduct2_f;

      /* Sqrt: '<S163>/Sqrt3' */
      Vehicle_B.Sqrt3_l = std::sqrt(Vehicle_B.Divide1_b);

      /* Product: '<S163>/Divide' */
      Vehicle_B.Divide_k = Vehicle_B.DotProduct3_p / Vehicle_B.Sqrt3_l;

      /* Bias: '<S163>/Bias' */
      Vehicle_B.Bias_b = Vehicle_B.Divide_k + -1.0;

      /* Abs: '<S163>/Abs' */
      Vehicle_B.Abs_o = std::abs(Vehicle_B.Bias_b);

      /* Bias: '<S163>/Bias1' */
      Vehicle_B.Bias1_m = Vehicle_B.Divide_k + 1.0;

      /* Abs: '<S163>/Abs1' */
      Vehicle_B.Abs1_l = std::abs(Vehicle_B.Bias1_m);

      /* If: '<S163>/If' */
      if (Vehicle_B.Abs_o < 1.0E-6) {
        /* Outputs for IfAction SubSystem: '<S163>/If Action Subsystem' incorporates:
         *  ActionPort: '<S169>/Action Port'
         */
        //Vehicle_IfActionSubsystem(&Vehicle_B.Merge_d);
        Vehicle_B.Merge_d = 1.0;
        /* End of Outputs for SubSystem: '<S163>/If Action Subsystem' */
      } 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;
        /* End of Outputs for SubSystem: '<S163>/If Action Subsystem1' */
      } 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 Outputs for SubSystem: '<S163>/If Action Subsystem2' */
      }

      /* 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' */
        Vehicle_B.jxi_k = 0.0 * Vehicle_B.Sum_k;
```

```
          /* Product: '<S168>/i x k' */
          Vehicle_B.ixk_a = 0.0 * Vehicle_B.Sum_n;

          /* Product: '<S168>/k x j' */
          Vehicle_B.kxj_c = -Vehicle_B.Sum_lm;

          /* Product: '<S167>/i x j' */
          Vehicle_B.ixj_o = 0.0 * Vehicle_B.Sum_lm;

          /* Product: '<S167>/k x i' */
          Vehicle_B.kxi_p = -Vehicle_B.Sum_k;

          /* Product: '<S167>/j x k' */
          Vehicle_B.jxk_er = 0.0 * Vehicle_B.Sum_n;

          /* Sum: '<S162>/Sum' */
          Vehicle_B.Sum_ff[0] = Vehicle_B.jxk_er - Vehicle_B.kxj_c;
          Vehicle_B.Sum_ff[1] = Vehicle_B.kxi_p - Vehicle_B.ixk_a;
          Vehicle_B.Sum_ff[2] = Vehicle_B.ixj_o - Vehicle_B.jxi_k;

          /* Switch: '<S136>/is 180deg Rot' */
          Vehicle_B.is180degRot[0] = Vehicle_B.Sum_ff[0];
          Vehicle_B.is180degRot[1] = Vehicle_B.Sum_ff[1];
          Vehicle_B.is180degRot[2] = Vehicle_B.Sum_ff[2];
        } else {
          /* Switch: '<S136>/is 180deg Rot' */
          Vehicle_B.is180degRot[0] = Vehicle_ConstB.Sum[0];
          Vehicle_B.is180degRot[1] = Vehicle_ConstB.Sum[1];
          Vehicle_B.is180degRot[2] = Vehicle_ConstB.Sum[2];
        }

        /* End of Switch: '<S136>/is 180deg Rot' */

        /* Product: '<S176>/Product1' */
        Vehicle_B.Product1_g = Vehicle_B.is180degRot[0] * Vehicle_B.is180degRot[0];

        /* Product: '<S176>/Product2' */
        Vehicle_B.Product2_g = Vehicle_B.is180degRot[1] * Vehicle_B.is180degRot[1];

        /* Product: '<S176>/Product3' */
        Vehicle_B.Product3_aj = Vehicle_B.is180degRot[2] * Vehicle_B.is180degRot[2];

        /* Sum: '<S176>/Sum' */
        Vehicle_B.Sum_c = ((Vehicle_B.Product_i2 + Vehicle_B.Product1_g) +
                           Vehicle_B.Product2_g) + Vehicle_B.Product3_aj;

        /* Sqrt: '<S175>/sqrt' */
        Vehicle_B.sqrt_h = std::sqrt(Vehicle_B.Sum_c);

        /* Product: '<S166>/Product' */
        Vehicle_B.Product_fv = Vehicle_B.Add2 / Vehicle_B.sqrt_h;

        /* Product: '<S166>/Product2' */
        Vehicle_B.Product2_a = Vehicle_B.is180degRot[1] / Vehicle_B.sqrt_h;

        /* Product: '<S155>/Product' */
        Vehicle_B.Product_l5 = Vehicle_B.Product_fv * Vehicle_B.Product_fv;

        /* Product: '<S166>/Product1' */
        Vehicle_B.Product1_n = Vehicle_B.is180degRot[0] / Vehicle_B.sqrt_h;

        /* Product: '<S155>/Product1' */
        Vehicle_B.Product1_lg = Vehicle_B.Product1_n * Vehicle_B.Product1_n;

        /* Product: '<S155>/Product2' */
        Vehicle_B.Product2_fh = Vehicle_B.Product2_a * Vehicle_B.Product2_a;

        /* Product: '<S166>/Product3' */
        Vehicle_B.Product3_pe = Vehicle_B.is180degRot[2] / Vehicle_B.sqrt_h;

        /* Product: '<S155>/Product3' */
        Vehicle_B.Product3_l0 = Vehicle_B.Product3_pe * Vehicle_B.Product3_pe;

        /* Sum: '<S155>/Sum' */
        Vehicle_B.Sum_n3 = ((Vehicle_B.Product_l5 + Vehicle_B.Product1_lg) +
                            Vehicle_B.Product2_fh) + Vehicle_B.Product3_l0;

        /* Sqrt: '<S154>/sqrt' */
        Vehicle_B.sqrt_mv = std::sqrt(Vehicle_B.Sum_n3);

        /* Product: '<S150>/Product2' */
        Vehicle_B.Product2_ke = Vehicle_B.Product2_a / Vehicle_B.sqrt_mv;

        /* Product: '<S151>/Product6' */
        Vehicle_B.Product6_e = Vehicle_B.Product2_ke * Vehicle_B.Product2_ke;

        /* Product: '<S150>/Product3' */
        Vehicle_B.Product3_cc = Vehicle_B.Product3_pe / Vehicle_B.sqrt_mv;

        /* Product: '<S151>/Product7' */
        Vehicle_B.Product7_a = Vehicle_B.Product3_cc * Vehicle_B.Product3_cc;

        /* Sum: '<S151>/Sum3' incorporates:
         *  Constant: '<S151>/Constant'
         */
        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;

        /* Product: '<S151>/Product8' */
        Vehicle_B.Product8_k = Vehicle_ConstB.Sum_d0[0] * Vehicle_B.Gain2_p;
```

```
        /* Product: '<S150>/Product1' */
        Vehicle_B.Product1_l0 = Vehicle_B.Product1_n / Vehicle_B.sqrt_mv;

        /* Product: '<S151>/Product' */
        Vehicle_B.Product_da = Vehicle_B.Product1_l0 * Vehicle_B.Product2_ke;

        /* Product: '<S150>/Product' */
        Vehicle_B.Product_m = Vehicle_B.Product_fv / Vehicle_B.sqrt_mv;

        /* Product: '<S151>/Product1' */
        Vehicle_B.Product1_dj = Vehicle_B.Product_m * Vehicle_B.Product3_cc;

        /* Sum: '<S151>/Sum1' */
        Vehicle_B.Sum1_n = Vehicle_B.Product_da + Vehicle_B.Product1_dj;

        /* Gain: '<S151>/Gain' */
        Vehicle_B.Gain_l = 2.0 * Vehicle_B.Sum1_n;

        /* Product: '<S151>/Product4' */
        Vehicle_B.Product4_c = Vehicle_B.Gain_l * Vehicle_ConstB.Sum_d0[1];

        /* Product: '<S151>/Product2' */
        Vehicle_B.Product2_gk = Vehicle_B.Product_m * Vehicle_B.Product2_ke;

        /* Product: '<S151>/Product3' */
        Vehicle_B.Product3_i = Vehicle_B.Product1_l0 * Vehicle_B.Product3_cc;

        /* Sum: '<S151>/Sum2' */
        Vehicle_B.Sum2_g = Vehicle_B.Product3_i - Vehicle_B.Product2_gk;

        /* Gain: '<S151>/Gain1' */
        Vehicle_B.Gain1_e = 2.0 * Vehicle_B.Sum2_g;

        /* Product: '<S151>/Product5' */
        Vehicle_B.Product5_o = Vehicle_B.Gain1_e * Vehicle_ConstB.Sum_d0[2];

        /* Sum: '<S151>/Sum' */
        Vehicle_B.Sum_oz = (Vehicle_B.Product8_k + Vehicle_B.Product4_c) +
          Vehicle_B.Product5_o;

        /* Product: '<S152>/Product' */
        Vehicle_B.Product_g = Vehicle_B.Product1_l0 * Vehicle_B.Product2_ke;

        /* Product: '<S152>/Product1' */
        Vehicle_B.Product1_ew = Vehicle_B.Product_m * Vehicle_B.Product3_cc;

        /* Sum: '<S152>/Sum1' */
        Vehicle_B.Sum1_m = Vehicle_B.Product_g - Vehicle_B.Product1_ew;

        /* Gain: '<S152>/Gain' */
        Vehicle_B.Gain_b = 2.0 * Vehicle_B.Sum1_m;

        /* Product: '<S152>/Product4' */
        Vehicle_B.Product4_a = Vehicle_ConstB.Sum_d0[0] * Vehicle_B.Gain_b;

        /* Product: '<S152>/Product6' */
        Vehicle_B.Product6_o = Vehicle_B.Product1_l0 * Vehicle_B.Product1_l0;

        /* Product: '<S152>/Product7' */
        Vehicle_B.Product7_k = Vehicle_B.Product3_cc * Vehicle_B.Product3_cc;

        /* Sum: '<S152>/Sum3' incorporates:
         *  Constant: '<S152>/Constant'
         */
        Vehicle_B.Sum3_bi = (0.5 - Vehicle_B.Product6_o) - Vehicle_B.Product7_k;

        /* Gain: '<S152>/Gain2' */
        Vehicle_B.Gain2_l = 2.0 * Vehicle_B.Sum3_bi;

        /* Product: '<S152>/Product8' */
        Vehicle_B.Product8_me = Vehicle_B.Gain2_l * Vehicle_ConstB.Sum_d0[1];

        /* Product: '<S152>/Product2' */
        Vehicle_B.Product2_fhq = Vehicle_B.Product_m * Vehicle_B.Product1_l0;

        /* Product: '<S152>/Product3' */
        Vehicle_B.Product3_n = Vehicle_B.Product2_ke * Vehicle_B.Product3_cc;

        /* Sum: '<S152>/Sum2' */
        Vehicle_B.Sum2_la = Vehicle_B.Product2_fhq + Vehicle_B.Product3_n;

        /* Gain: '<S152>/Gain1' */
        Vehicle_B.Gain1_ox = 2.0 * Vehicle_B.Sum2_la;

        /* Product: '<S152>/Product5' */
        Vehicle_B.Product5_l = Vehicle_B.Gain1_ox * Vehicle_ConstB.Sum_d0[2];

        /* Sum: '<S152>/Sum' */
        Vehicle_B.Sum_oa = (Vehicle_B.Product4_a + Vehicle_B.Product8_me) +
          Vehicle_B.Product5_l;

        /* Product: '<S153>/Product' */
        Vehicle_B.Product_ew = Vehicle_B.Product1_l0 * Vehicle_B.Product3_cc;

        /* Product: '<S153>/Product1' */
        Vehicle_B.Product1_m = Vehicle_B.Product_m * Vehicle_B.Product2_ke;

        /* Sum: '<S153>/Sum1' */
        Vehicle_B.Sum1_h = Vehicle_B.Product_ew + Vehicle_B.Product1_m;

        /* Gain: '<S153>/Gain' */
        Vehicle_B.Gain_kp = 2.0 * Vehicle_B.Sum1_h;

        /* Product: '<S153>/Product4' */
```

```
        Vehicle_B.Product4_g = Vehicle_ConstB.Sum_d0[0] * Vehicle_B.Gain_kp;

        /* Product: '<S153>/Product2' */
        Vehicle_B.Product2_gh = Vehicle_B.Product_m * Vehicle_B.Product1_l0;

        /* Product: '<S153>/Product3' */
        Vehicle_B.Product3_k4 = Vehicle_B.Product2_ke * Vehicle_B.Product3_cc;

        /* Sum: '<S153>/Sum2' */
        Vehicle_B.Sum2_i = Vehicle_B.Product3_k4 - Vehicle_B.Product2_gh;

        /* Gain: '<S153>/Gain1' */
        Vehicle_B.Gain1_f = 2.0 * Vehicle_B.Sum2_i;

        /* Product: '<S153>/Product5' */
        Vehicle_B.Product5_ij = Vehicle_B.Gain1_f * Vehicle_ConstB.Sum_d0[1];

        /* Product: '<S153>/Product6' */
        Vehicle_B.Product6_c = Vehicle_B.Product1_l0 * Vehicle_B.Product1_l0;

        /* Product: '<S153>/Product7' */
        Vehicle_B.Product7_a4 = Vehicle_B.Product2_ke * Vehicle_B.Product2_ke;

        /* Sum: '<S153>/Sum3' incorporates:
         *  Constant: '<S153>/Constant'
         */
        Vehicle_B.Sum3_d = (0.5 - Vehicle_B.Product6_c) - Vehicle_B.Product7_a4;

        /* Gain: '<S153>/Gain2' */
        Vehicle_B.Gain2_n = 2.0 * Vehicle_B.Sum3_d;

        /* Product: '<S153>/Product8' */
        Vehicle_B.Product8_f = Vehicle_B.Gain2_n * Vehicle_ConstB.Sum_d0[2];

        /* Sum: '<S153>/Sum' */
        Vehicle_B.Sum_nx = (Vehicle_B.Product4_g + Vehicle_B.Product5_ij) +
          Vehicle_B.Product8_f;

        /* SignalConversion generated from: '<S178>/Dot Product1' */
        Vehicle_B.TmpSignalConversionAtDotProdu_k[0] = Vehicle_B.Sum_oz;
        Vehicle_B.TmpSignalConversionAtDotProdu_k[1] = Vehicle_B.Sum_oa;
        Vehicle_B.TmpSignalConversionAtDotProdu_k[2] = Vehicle_B.Sum_nx;

        /* Product: '<S200>/j x k' */
        Vehicle_B.jxk_b = Vehicle_B.Sum_lm * Vehicle_B.Sum_kb;

        /* Product: '<S200>/k x i' */
        Vehicle_B.kxi_b = Vehicle_B.Sum_n * Vehicle_B.Sum_kk;

        /* Product: '<S200>/i x j' */
        Vehicle_B.ixj_h = Vehicle_B.Sum_k * Vehicle_B.Sum_ln;

        /* Product: '<S201>/k x j' */
        Vehicle_B.kxj_l = Vehicle_B.Sum_n * Vehicle_B.Sum_ln;

        /* Product: '<S201>/i x k' */
        Vehicle_B.ixk_e = Vehicle_B.Sum_k * Vehicle_B.Sum_kb;

        /* Product: '<S201>/j x i' */
        Vehicle_B.jxi_n = Vehicle_B.Sum_lm * Vehicle_B.Sum_kk;

        /* Sum: '<S198>/Sum' */
        Vehicle_B.Sum_m[0] = Vehicle_B.jxk_b - Vehicle_B.kxj_l;
        Vehicle_B.Sum_m[1] = Vehicle_B.kxi_b - Vehicle_B.ixk_e;
        Vehicle_B.Sum_m[2] = Vehicle_B.ixj_h - Vehicle_B.jxi_n;

        /* Product: '<S202>/j x k' */
        Vehicle_B.jxk_e = Vehicle_B.Sum_m[1] * Vehicle_B.Sum_n;

        /* Product: '<S202>/k x i' */
        Vehicle_B.kxi_es = Vehicle_B.Sum_m[2] * Vehicle_B.Sum_k;

        /* Product: '<S202>/i x j' */
        Vehicle_B.ixj_n = Vehicle_B.Sum_m[0] * Vehicle_B.Sum_lm;

        /* Product: '<S203>/k x j' */
        Vehicle_B.kxj_j = Vehicle_B.Sum_m[2] * Vehicle_B.Sum_lm;

        /* Product: '<S203>/i x k' */
        Vehicle_B.ixk_j = Vehicle_B.Sum_m[0] * Vehicle_B.Sum_n;

        /* Product: '<S203>/j x i' */
        Vehicle_B.jxi_f = Vehicle_B.Sum_m[1] * Vehicle_B.Sum_k;

        /* Sum: '<S199>/Sum' */
        Vehicle_B.Sum_pb[0] = Vehicle_B.jxk_e - Vehicle_B.kxj_j;
        Vehicle_B.Sum_pb[1] = Vehicle_B.kxi_es - Vehicle_B.ixk_j;
        Vehicle_B.Sum_pb[2] = Vehicle_B.ixj_n - Vehicle_B.jxi_f;

        /* Product: '<S161>/Product' */
        Vehicle_B.Product_dj = Vehicle_B.Product_fv * Vehicle_B.Product_fv;

        /* Product: '<S161>/Product1' */
        Vehicle_B.Product1_ig = Vehicle_B.Product1_n * Vehicle_B.Product1_n;

        /* Product: '<S161>/Product2' */
        Vehicle_B.Product2_ad = Vehicle_B.Product2_a * Vehicle_B.Product2_a;

        /* Product: '<S161>/Product3' */
        Vehicle_B.Product3_hf = Vehicle_B.Product3_pe * Vehicle_B.Product3_pe;

        /* Sum: '<S161>/Sum' */
        Vehicle_B.Sum_gt = ((Vehicle_B.Product_dj + Vehicle_B.Product1_ig) +
                            Vehicle_B.Product2_ad) + Vehicle_B.Product3_hf;
```

```
    /* Sqrt: '<S160>/sqrt' */
    Vehicle_B.sqrt_i = std::sqrt(Vehicle_B.Sum_gt);

    /* Product: '<S156>/Product2' */
    Vehicle_B.Product2_nx = Vehicle_B.Product2_a / Vehicle_B.sqrt_i;

    /* Product: '<S157>/Product6' */
    Vehicle_B.Product6_a = Vehicle_B.Product2_nx * Vehicle_B.Product2_nx;

    /* Product: '<S156>/Product3' */
    Vehicle_B.Product3_ce = Vehicle_B.Product3_pe / Vehicle_B.sqrt_i;

    /* Product: '<S157>/Product7' */
    Vehicle_B.Product7_ha = Vehicle_B.Product3_ce * Vehicle_B.Product3_ce;

    /* Sum: '<S157>/Sum3' incorporates:
     *  Constant: '<S157>/Constant'
     */
    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;

    /* Product: '<S157>/Product8' */
    Vehicle_B.Product8_p = Vehicle_B.Sum_pb[0] * Vehicle_B.Gain2_om;

    /* Product: '<S156>/Product1' */
    Vehicle_B.Product1_a = Vehicle_B.Product1_n / Vehicle_B.sqrt_i;

    /* Product: '<S157>/Product' */
    Vehicle_B.Product_mi = Vehicle_B.Product1_a * Vehicle_B.Product2_nx;

    /* Product: '<S156>/Product' */
    Vehicle_B.Product_pb = Vehicle_B.Product_fv / Vehicle_B.sqrt_i;

    /* Product: '<S157>/Product1' */
    Vehicle_B.Product1_lr = Vehicle_B.Product_pb * Vehicle_B.Product3_ce;

    /* Sum: '<S157>/Sum1' */
    Vehicle_B.Sum1_a = Vehicle_B.Product_mi + Vehicle_B.Product1_lr;

    /* Gain: '<S157>/Gain' */
    Vehicle_B.Gain_p = 2.0 * Vehicle_B.Sum1_a;

    /* Product: '<S157>/Product4' */
    Vehicle_B.Product4_i = Vehicle_B.Gain_p * Vehicle_B.Sum_pb[1];

    /* Product: '<S157>/Product2' */
    Vehicle_B.Product2_nz = Vehicle_B.Product_pb * Vehicle_B.Product2_nx;

    /* Product: '<S157>/Product3' */
    Vehicle_B.Product3_lu = Vehicle_B.Product1_a * Vehicle_B.Product3_ce;

    /* Sum: '<S157>/Sum2' */
    Vehicle_B.Sum2_m = Vehicle_B.Product3_lu - Vehicle_B.Product2_nz;

    /* Gain: '<S157>/Gain1' */
    Vehicle_B.Gain1_c = 2.0 * Vehicle_B.Sum2_m;

    /* Product: '<S157>/Product5' */
    Vehicle_B.Product5_jx = Vehicle_B.Gain1_c * Vehicle_B.Sum_pb[2];

    /* Sum: '<S157>/Sum' */
    Vehicle_B.Sum_g5 = (Vehicle_B.Product8_p + Vehicle_B.Product4_i) +
      Vehicle_B.Product5_jx;

    /* Product: '<S158>/Product' */
    Vehicle_B.Product_pe = Vehicle_B.Product1_a * Vehicle_B.Product2_nx;

    /* Product: '<S158>/Product1' */
    Vehicle_B.Product1_mu = Vehicle_B.Product_pb * Vehicle_B.Product3_ce;

    /* Sum: '<S158>/Sum1' */
    Vehicle_B.Sum1_k = Vehicle_B.Product_pe - Vehicle_B.Product1_mu;

    /* Gain: '<S158>/Gain' */
    Vehicle_B.Gain_ay = 2.0 * Vehicle_B.Sum1_k;

    /* Product: '<S158>/Product4' */
    Vehicle_B.Product4_ok = Vehicle_B.Sum_pb[0] * Vehicle_B.Gain_ay;

    /* Product: '<S158>/Product6' */
    Vehicle_B.Product6_ld = Vehicle_B.Product1_a * Vehicle_B.Product1_a;

    /* Product: '<S158>/Product7' */
    Vehicle_B.Product7_ei = Vehicle_B.Product3_ce * Vehicle_B.Product3_ce;

    /* Sum: '<S158>/Sum3' incorporates:
     *  Constant: '<S158>/Constant'
     */
    Vehicle_B.Sum3_h = (0.5 - Vehicle_B.Product6_ld) - Vehicle_B.Product7_ei;

    /* Gain: '<S158>/Gain2' */
    Vehicle_B.Gain2_l2 = 2.0 * Vehicle_B.Sum3_h;

    /* Product: '<S158>/Product8' */
    Vehicle_B.Product8_d = Vehicle_B.Gain2_l2 * Vehicle_B.Sum_pb[1];

    /* Product: '<S158>/Product2' */
    Vehicle_B.Product2_ea = Vehicle_B.Product_pb * Vehicle_B.Product1_a;

    /* Product: '<S158>/Product3' */
    Vehicle_B.Product3_ll = Vehicle_B.Product2_nx * Vehicle_B.Product3_ce;
```

```c
    /* Sum: '<S158>/Sum2' */
    Vehicle_B.Sum2_mr = Vehicle_B.Product2_ea + Vehicle_B.Product3_ll;

    /* Gain: '<S158>/Gain1' */
    Vehicle_B.Gain1_n = 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: '<S159>/Product' */
    Vehicle_B.Product_am = Vehicle_B.Product1_a * Vehicle_B.Product3_ce;

    /* Product: '<S159>/Product1' */
    Vehicle_B.Product1_bt = Vehicle_B.Product_pb * Vehicle_B.Product2_nx;

    /* Sum: '<S159>/Sum1' */
    Vehicle_B.Sum1_hk = Vehicle_B.Product_am + Vehicle_B.Product1_bt;

    /* Gain: '<S159>/Gain' */
    Vehicle_B.Gain_g = 2.0 * Vehicle_B.Sum1_hk;

    /* Product: '<S159>/Product4' */
    Vehicle_B.Product4_o5 = Vehicle_B.Sum_pb[0] * Vehicle_B.Gain_g;

    /* Product: '<S159>/Product2' */
    Vehicle_B.Product2_cc = Vehicle_B.Product_pb * Vehicle_B.Product1_a;

    /* Product: '<S159>/Product3' */
    Vehicle_B.Product3_cd = Vehicle_B.Product2_nx * Vehicle_B.Product3_ce;

    /* Sum: '<S159>/Sum2' */
    Vehicle_B.Sum2_o = Vehicle_B.Product3_cd - Vehicle_B.Product2_cc;

    /* Gain: '<S159>/Gain1' */
    Vehicle_B.Gain1_ni = 2.0 * Vehicle_B.Sum2_o;

    /* Product: '<S159>/Product5' */
    Vehicle_B.Product5_n2 = Vehicle_B.Gain1_ni * Vehicle_B.Sum_pb[1];

    /* Product: '<S159>/Product6' */
    Vehicle_B.Product6_d5 = Vehicle_B.Product1_a * Vehicle_B.Product1_a;

    /* Product: '<S159>/Product7' */
    Vehicle_B.Product7_em = Vehicle_B.Product2_nx * Vehicle_B.Product2_nx;

    /* Sum: '<S159>/Sum3' incorporates:
     *  Constant: '<S159>/Constant'
     */
    Vehicle_B.Sum3_jh = (0.5 - Vehicle_B.Product6_d5) - Vehicle_B.Product7_em;

    /* Gain: '<S159>/Gain2' */
    Vehicle_B.Gain2_ja = 2.0 * Vehicle_B.Sum3_jh;

    /* Product: '<S159>/Product8' */
    Vehicle_B.Product8_j = Vehicle_B.Gain2_ja * Vehicle_B.Sum_pb[2];

    /* Sum: '<S159>/Sum' */
    Vehicle_B.Sum_pw = (Vehicle_B.Product4_o5 + Vehicle_B.Product5_n2) +
      Vehicle_B.Product8_j;

    /* SignalConversion generated from: '<S178>/Dot Product2' */
    Vehicle_B.TmpSignalConversionAtDotProdu_j[0] = Vehicle_B.Sum_g5;
    Vehicle_B.TmpSignalConversionAtDotProdu_j[1] = Vehicle_B.Sum_gz;
    Vehicle_B.TmpSignalConversionAtDotProdu_j[2] = Vehicle_B.Sum_pw;

    /* DotProduct: '<S178>/Dot Product3' */
    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_j[0];
    tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_j[0];
    tmpForInput_0 = tmpForInput_idx_1 * tmpForInput_idx_0;

    /* DotProduct: '<S178>/Dot Product3' */
    tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProdu_k[1];
    tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_j[1];
    tmpForInput_idx_2 += 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_j[1];
    tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_j[1];
    tmpForInput_0 += tmpForInput_idx_1 * tmpForInput_idx_0;

    /* DotProduct: '<S178>/Dot Product3' */
    tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProdu_k[2];
    tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_j[2];
    tmpForInput_idx_2 += tmpForInput_idx_1 * tmpForInput_idx_0;
```

```
      /* DotProduct: '<S178>/Dot Product1' */
      tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProdu_k[2];
      tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_k[2];
      tmpForInput += tmpForInput_idx_1 * tmpForInput_idx_0;

      /* DotProduct: '<S178>/Dot Product2' */
      tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProdu_j[2];
      tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_j[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>/Sqrt3' */
      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>/Bias' */
      Vehicle_B.Bias_h = Vehicle_B.Divide_c + -1.0;

      /* Abs: '<S178>/Abs' */
      Vehicle_B.Abs_m = std::abs(Vehicle_B.Bias_h);

      /* Bias: '<S178>/Bias1' */
      Vehicle_B.Bias1_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 IfAction SubSystem: '<S178>/If Action Subsystem' incorporates:
         *  ActionPort: '<S184>/Action Port'
         */
        //Vehicle_IfActionSubsystem(&Vehicle_B.Merge_l);
          Vehicle_B.Merge_l = 1.0;
        /* End of Outputs for SubSystem: '<S178>/If Action Subsystem' */
      } else if (Vehicle_B.Abs1_i < 1.0E-6) {
        /* Outputs for IfAction SubSystem: '<S178>/If Action Subsystem1' incorporates:
         *  ActionPort: '<S185>/Action Port'
         */
        //Vehicle_IfActionSubsystem1(&Vehicle_B.Merge_l);
          Vehicle_B.Merge_l = -1.0;
        /* End of Outputs for SubSystem: '<S178>/If Action Subsystem1' */
      } else {
        /* Outputs for IfAction SubSystem: '<S178>/If Action Subsystem2' incorporates:
         *  ActionPort: '<S186>/Action Port'
         */
       // Vehicle_IfActionSubsystem2(&Vehicle_B.Merge_l);
          Vehicle_B.Merge_l = 0.0;
        /* End of Outputs for SubSystem: '<S178>/If Action 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:
         *  Constant: '<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 i' */
          Vehicle_B.jxi_no = Vehicle_B.Sum_oa * Vehicle_B.Sum_g5;

          /* Product: '<S183>/i x k' */
          Vehicle_B.ixk_b = Vehicle_B.Sum_oz * Vehicle_B.Sum_pw;

          /* Product: '<S183>/k x j' */
          Vehicle_B.kxj_n = Vehicle_B.Sum_nx * Vehicle_B.Sum_gz;

          /* Product: '<S182>/i x j' */
          Vehicle_B.ixj_l = Vehicle_B.Sum_oz * Vehicle_B.Sum_gz;

          /* Product: '<S182>/k x i' */
          Vehicle_B.kxi_o = Vehicle_B.Sum_nx * Vehicle_B.Sum_g5;

          /* Product: '<S182>/j x k' */
          Vehicle_B.jxk_d = Vehicle_B.Sum_oa * Vehicle_B.Sum_pw;

          /* Sum: '<S177>/Sum' */
          Vehicle_B.Sum_b[0] = Vehicle_B.jxk_d - Vehicle_B.kxj_n;
          Vehicle_B.Sum_b[1] = Vehicle_B.kxi_o - Vehicle_B.ixk_b;
          Vehicle_B.Sum_b[2] = Vehicle_B.ixj_l - Vehicle_B.jxi_no;
```

```
          /* Switch: '<S137>/is 180deg Rot' */
          Vehicle_B.is180degRot_l[0] = Vehicle_B.Sum_b[0];
          Vehicle_B.is180degRot_l[1] = Vehicle_B.Sum_b[1];
          Vehicle_B.is180degRot_l[2] = Vehicle_B.Sum_b[2];
        } else {
          /* RelationalOperator: '<S180>/x>z' */
          Vehicle_B.xz = (Vehicle_B.Sum_oz > 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_oz;
            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.ixk_k = Vehicle_B.Sum_oz * 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_oz * Vehicle_B.Switch3[1];

          /* Product: '<S188>/k x i' */
          Vehicle_B.kxi_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;
          Vehicle_B.Sum_pd[1] = Vehicle_B.kxi_bj - Vehicle_B.ixk_k;
          Vehicle_B.Sum_pd[2] = Vehicle_B.ixj_k - Vehicle_B.jxi_m;

          /* Switch: '<S137>/is 180deg Rot' */
          Vehicle_B.is180degRot_l[0] = Vehicle_B.Sum_pd[0];
          Vehicle_B.is180degRot_l[1] = Vehicle_B.Sum_pd[1];
          Vehicle_B.is180degRot_l[2] = Vehicle_B.Sum_pd[2];
        }

        /* End of Switch: '<S137>/is 180deg Rot' */

        /* Product: '<S191>/Product3' */
        Vehicle_B.Product3_dz = Vehicle_B.is180degRot_l[2] *
          Vehicle_B.is180degRot_l[2];

        /* Product: '<S191>/Product2' */
        Vehicle_B.Product2_eq = Vehicle_B.is180degRot_l[1] *
          Vehicle_B.is180degRot_l[1];

        /* Product: '<S191>/Product1' */
        Vehicle_B.Product1_aa = Vehicle_B.is180degRot_l[0] *
          Vehicle_B.is180degRot_l[0];

        /* DotProduct: '<S137>/Dot Product2' */
        tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProdu_j[0];
        tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_j[0];
        tmpForInput_idx_2 = tmpForInput_idx_1 * tmpForInput_idx_0;

        /* DotProduct: '<S137>/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: '<S137>/Dot Product3' */
        tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProdu_k[0];
        tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_j[0];
        tmpForInput_0 = tmpForInput_idx_1 * tmpForInput_idx_0;

        /* DotProduct: '<S137>/Dot Product2' */
        tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProdu_j[1];
        tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_j[1];
        tmpForInput_idx_2 += tmpForInput_idx_1 * tmpForInput_idx_0;

        /* DotProduct: '<S137>/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: '<S137>/Dot Product3' */
        tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProdu_k[1];
```

```
    tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_j[1];
    tmpForInput_0 += tmpForInput_idx_1 * tmpForInput_idx_0;

    /* DotProduct: '<S137>/Dot Product2' */
    tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProdu_j[2];
    tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_j[2];
    tmpForInput_idx_2 += tmpForInput_idx_1 * tmpForInput_idx_0;

    /* DotProduct: '<S137>/Dot Product1' */
    tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProdu_k[2];
    tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_k[2];
    tmpForInput += tmpForInput_idx_1 * tmpForInput_idx_0;

    /* DotProduct: '<S137>/Dot Product3' */
    tmpForInput_idx_1 = Vehicle_B.TmpSignalConversionAtDotProdu_k[2];
    tmpForInput_idx_0 = Vehicle_B.TmpSignalConversionAtDotProdu_j[2];
    tmpForInput_0 += tmpForInput_idx_1 * tmpForInput_idx_0;

    /* DotProduct: '<S137>/Dot Product2' */
    Vehicle_B.DotProduct2_n = tmpForInput_idx_2;

    /* DotProduct: '<S137>/Dot Product1' */
    Vehicle_B.DotProduct1_h = tmpForInput;

    /* Product: '<S137>/Divide1' */
    Vehicle_B.Divide1_h = Vehicle_B.DotProduct1_h * Vehicle_B.DotProduct2_n;

    /* Sqrt: '<S137>/Sqrt3' */
    Vehicle_B.Sqrt3_mq = std::sqrt(Vehicle_B.Divide1_h);

    /* DotProduct: '<S137>/Dot Product3' */
    Vehicle_B.DotProduct3_e = tmpForInput_0;

    /* Sum: '<S137>/Add2' */
    Vehicle_B.Add2_k = Vehicle_B.DotProduct3_e + Vehicle_B.Sqrt3_mq;

    /* Product: '<S191>/Product' */
    Vehicle_B.Product_er = Vehicle_B.Add2_k * Vehicle_B.Add2_k;

    /* Sum: '<S191>/Sum' */
    Vehicle_B.Sum_e = ((Vehicle_B.Product_er + Vehicle_B.Product1_aa) +
                       Vehicle_B.Product2_eq) + Vehicle_B.Product3_dz;

    /* Sqrt: '<S190>/sqrt' */
    Vehicle_B.sqrt_j = std::sqrt(Vehicle_B.Sum_e);

    /* Product: '<S181>/Product3' */
    Vehicle_B.Product3_lzz = Vehicle_B.is180degRot_l[2] / Vehicle_B.sqrt_j;

    /* Product: '<S146>/Product3' */
    Vehicle_B.Product3_ay = Vehicle_B.Product3_pe * Vehicle_B.Product3_lzz;

    /* Product: '<S181>/Product2' */
    Vehicle_B.Product2_ep = Vehicle_B.is180degRot_l[1] / Vehicle_B.sqrt_j;

    /* Product: '<S146>/Product2' */
    Vehicle_B.Product2_bc = Vehicle_B.Product2_a * Vehicle_B.Product2_ep;

    /* Product: '<S181>/Product1' */
    Vehicle_B.Product1_ge = Vehicle_B.is180degRot_l[0] / Vehicle_B.sqrt_j;

    /* Product: '<S146>/Product1' */
    Vehicle_B.Product1_fh = Vehicle_B.Product1_n * Vehicle_B.Product1_ge;

    /* Product: '<S181>/Product' */
    Vehicle_B.Product_ey = Vehicle_B.Add2_k / Vehicle_B.sqrt_j;

    /* Product: '<S146>/Product' */
    Vehicle_B.Product_d0 = Vehicle_B.Product_fv * Vehicle_B.Product_ey;

    /* Sum: '<S146>/Sum' */
    Vehicle_B.Sum_ki = ((Vehicle_B.Product_d0 - Vehicle_B.Product1_fh) -
                        Vehicle_B.Product2_bc) - Vehicle_B.Product3_ay;

    /* Product: '<S147>/Product3' */
    Vehicle_B.Product3_aw = Vehicle_B.Product3_pe * Vehicle_B.Product2_ep;

    /* Product: '<S147>/Product2' */
    Vehicle_B.Product2_pu = Vehicle_B.Product2_a * Vehicle_B.Product3_lzz;

    /* Product: '<S147>/Product1' */
    Vehicle_B.Product1_b0 = Vehicle_B.Product1_n * Vehicle_B.Product_ey;

    /* Product: '<S147>/Product' */
    Vehicle_B.Product_cq = Vehicle_B.Product_fv * Vehicle_B.Product1_ge;

    /* Sum: '<S147>/Sum' */
    Vehicle_B.Sum_oq = ((Vehicle_B.Product_cq + Vehicle_B.Product1_b0) +
                        Vehicle_B.Product2_pu) - Vehicle_B.Product3_aw;

    /* Product: '<S148>/Product3' */
    Vehicle_B.Product3_h5 = Vehicle_B.Product3_pe * Vehicle_B.Product1_ge;

    /* Product: '<S148>/Product2' */
    Vehicle_B.Product2_i = Vehicle_B.Product2_a * Vehicle_B.Product_ey;

    /* Product: '<S148>/Product1' */
    Vehicle_B.Product1_ai = Vehicle_B.Product1_n * Vehicle_B.Product3_lzz;

    /* Product: '<S148>/Product' */
    Vehicle_B.Product_cf = Vehicle_B.Product_fv * Vehicle_B.Product2_ep;

    /* Sum: '<S148>/Sum' */
    Vehicle_B.Sum_kc = ((Vehicle_B.Product_cf - Vehicle_B.Product1_ai) +
```

```
                            Vehicle_B.Product2_i) + Vehicle_B.Product3_h5;

          /* Product: '<S149>/Product3' */
          Vehicle_B.Product3_c1 = Vehicle_B.Product3_pe * Vehicle_B.Product_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_lzz;

          /* Sum: '<S149>/Sum' */
          Vehicle_B.Sum_n1 = ((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_oq;
          Vehicle_B.Switch_c[2] = Vehicle_B.Sum_kc;
          Vehicle_B.Switch_c[3] = Vehicle_B.Sum_n1;
        }

        /* 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;

        /* Sqrt: '<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' */
        Vehicle_B.Product2_j = Vehicle_B.Switch_c[2] / Vehicle_B.sqrt_mo;

        /* Product: '<S267>/Product3' */
        Vehicle_B.Product3_lz = Vehicle_B.Switch_c[3] / Vehicle_B.sqrt_mo;

        /* 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.VectorConcatenate_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 Subsystem' incorporates:
           *  ActionPort: '<S269>/Action Port'
           */
          /* Merge: '<S268>/Merge' incorporates:
           *  Constant: '<S269>/Constant'
           */
          Vehicle_B.Merge_k = 1.0;

          /* End of Outputs for SubSystem: '<S268>/If Action Subsystem' */
        } else if (Vehicle_B.fcn3_o < -1.0) {
          /* Outputs for IfAction SubSystem: '<S268>/If Action Subsystem1' incorporates:
           *  ActionPort: '<S270>/Action Port'
           */
          /* Merge: '<S268>/Merge' incorporates:
           *  Constant: '<S270>/Constant'
           */
          Vehicle_B.Merge_k = 1.0;

          /* End of Outputs for SubSystem: '<S268>/If Action Subsystem1' */
        } 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' */
```

```
        }

        /* End of If: '<S268>/If' */

        /* Trigonometry: '<S266>/trigFcn' */
        tmpForInput_idx_0 = Vehicle_B.Merge_k;
        if (tmpForInput_idx_0 > 1.0) {
          tmpForInput_idx_0 = 1.0;
        } else {
          if (tmpForInput_idx_0 < -1.0) {
            tmpForInput_idx_0 = -1.0;
          }
        }

        Vehicle_B.VectorConcatenate_a[1] = std::asin(tmpForInput_idx_0);

        /* End of Trigonometry: '<S266>/trigFcn' */

        /* Fcn: '<S128>/fcn4' */
        Vehicle_B.fcn4_n = (Vehicle_B.Product1_o * Vehicle_B.Product2_j -
                            Vehicle_B.Product_jc * Vehicle_B.Product3_lz) * -2.0;

        /* Fcn: '<S128>/fcn5' */
        Vehicle_B.fcn5_g = ((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 Function3' */
        Vehicle_B.VectorConcatenate_a[2] = 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>/UD' */
        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] = 1.00000000000000E-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>/UD' */
        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'
         *
         * 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[1] = Vehicle_B.DerivativeGain_k[1];

        /* Delay: '<S230>/UD' */
        Vehicle_B.UD_l[1] = Vehicle_DW.UD_DSTATE_d[1];

        /* Sum: '<S230>/Diff' */
        Vehicle_B.Diff_e[1] = Vehicle_B.Tsamp_f[1] - Vehicle_B.UD_l[1];

        /* Gain: '<S234>/Integral Gain' */
        Vehicle_B.IntegralGain_p[1] = 2.0E-10 * Vehicle_B.VectorConcatenate_a[1];

        /* DiscreteIntegrator: '<S237>/Integrator' */
        Vehicle_B.Integrator_k[1] = Vehicle_DW.Integrator_DSTATE_f[1];

        /* Gain: '<S242>/Proportional Gain' */
        Vehicle_B.ProportionalGain_h[1] = 1.00000000000000E-4 *
```

```
          Vehicle_B.VectorConcatenate_a[1];

        /* Merge: '<S1>/Merge' incorporates:
         *  Sum: '<S246>/Sum'
         */
        Vehicle_B.Merge[1] = (Vehicle_B.ProportionalGain_h[1] +
                              Vehicle_B.Integrator_k[1]) + Vehicle_B.Diff_e[1];

        /* Update for Delay: '<S230>/UD' */
        Vehicle_DW.UD_DSTATE_d[1] = Vehicle_B.Tsamp_f[1];

        /* Update for DiscreteIntegrator: '<S237>/Integrator' */
        Vehicle_DW.Integrator_DSTATE_f[1] += Vehicle_B.IntegralGain_p[1];

        /* Gain: '<S229>/Derivative Gain' */
        Vehicle_B.DerivativeGain_k[2] = 0.02 * Vehicle_B.VectorConcatenate_a[2];

        /* 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[2] = Vehicle_B.DerivativeGain_k[2];

        /* Delay: '<S230>/UD' */
        Vehicle_B.UD_l[2] = Vehicle_DW.UD_DSTATE_d[2];

        /* Sum: '<S230>/Diff' */
        Vehicle_B.Diff_e[2] = Vehicle_B.Tsamp_f[2] - Vehicle_B.UD_l[2];

        /* Gain: '<S234>/Integral Gain' */
        Vehicle_B.IntegralGain_p[2] = 2.0E-10 * Vehicle_B.VectorConcatenate_a[2];

        /* DiscreteIntegrator: '<S237>/Integrator' */
        Vehicle_B.Integrator_k[2] = Vehicle_DW.Integrator_DSTATE_f[2];

        /* Gain: '<S242>/Proportional Gain' */
        Vehicle_B.ProportionalGain_h[2] = 1.00000000000000E-4 *
          Vehicle_B.VectorConcatenate_a[2];

        /* Merge: '<S1>/Merge' incorporates:
         *  Sum: '<S246>/Sum'
         */
        Vehicle_B.Merge[2] = (Vehicle_B.ProportionalGain_h[2] +
                              Vehicle_B.Integrator_k[2]) + Vehicle_B.Diff_e[2];

        /* Update for Delay: '<S230>/UD' */
        Vehicle_DW.UD_DSTATE_d[2] = Vehicle_B.Tsamp_f[2];

        /* Update for DiscreteIntegrator: '<S237>/Integrator' */
        Vehicle_DW.Integrator_DSTATE_f[2] += Vehicle_B.IntegralGain_p[2];

        /* End of Outputs for SubSystem: '<S1>/SunPointing Mode' */
}
```

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 : 9.4 (R2020b) 29-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
 */

#ifndef 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

/* Block signals (default storage) */
typedef struct {
#ifdef RAD_HARD
      HardenedData::TripleData<real_T>    norm_b;

                 HardenedData::TripleData<real_T>    Merge[3];                        /* '<S1>/Merge' */
                 HardenedData::TripleData<real_T>    jxk;                             /* '<S20>/j x k' */
                 HardenedData::TripleData<real_T>    kxi;                             /* '<S20>/k x i' */
                 HardenedData::TripleData<real_T>    ixj;                             /* '<S20>/i x j' */
                 HardenedData::TripleData<real_T>    kxj;                             /* '<S21>/k x j' */
                 HardenedData::TripleData<real_T>    ixk;                             /* '<S21>/i x k' */
                 HardenedData::TripleData<real_T>    jxi;                             /* '<S21>/j x i' */
                 HardenedData::TripleData<real_T>    Sum[3];                          /* '<S13>/Sum' */
                 HardenedData::TripleData<real_T>    zr[3];                           /* '<S7>/z = -r' */
                 HardenedData::TripleData<real_T>    ixj_f;                           /* '<S18>/i x j' */
                 HardenedData::TripleData<real_T>    jxk_o;                           /* '<S18>/j x k' */
                 HardenedData::TripleData<real_T>    kxi_e;                           /* '<S18>/k x i' */
                 HardenedData::TripleData<real_T>    ixk_g;                           /* '<S19>/i x k' */
                 HardenedData::TripleData<real_T>    jxi_l;                           /* '<S19>/j x i' */
                 HardenedData::TripleData<real_T>    kxj_a;                           /* '<S19>/k x j' */
                 HardenedData::TripleData<real_T>    Sum_l[3];                        /* '<S12>/Sum' */
                 HardenedData::TripleData<real_T>    DotProduct;                      /* '<S15>/Dot Product' */
                 HardenedData::SingleData<real_T>    SumofElements;                   /* '<S15>/Sum of Elements' */
                 HardenedData::SingleData<real_T>    MathFunction1;                   /* '<S15>/Math Function1' */
                 HardenedData::SingleData<real_T>    DotProduct_p;                    /* '<S16>/Dot Product' */
                 HardenedData::SingleData<real_T>    SumofElements_c;                 /* '<S16>/Sum of Elements' */
                 HardenedData::SingleData<real_T>    MathFunction1_b;                 /* '<S16>/Math Function1' */
                 HardenedData::SingleData<real_T>    DotProduct_b;                    /* '<S17>/Dot Product' */
                 HardenedData::SingleData<real_T>    SumofElements_l;                 /* '<S17>/Sum of Elements' */
                 HardenedData::SingleData<real_T>    MathFunction1_m;                 /* '<S17>/Math Function1' */
                 HardenedData::SingleData<real_T>    ORFtoECEF[9];                    /* '<S7>/ORF to ECEF' */
                 HardenedData::SingleData<real_T>    ECEFtoORF[9];                    /* '<S7>/ECEF to ORF' */
                 HardenedData::TripleData<real_T>    Add;                             /* '<S25>/Add' */
                 HardenedData::TripleData<real_T>    Merge_i[4];                      /* '<S14>/Merge' */
                 HardenedData::TripleData<real_T>    Product;                         /* '<S111>/Product' */
                 HardenedData::TripleData<real_T>    Product1;                        /* '<S111>/Product1' */
                 HardenedData::TripleData<real_T>    Product2;                        /* '<S111>/Product2' */
                 HardenedData::TripleData<real_T>    Product3;                        /* '<S111>/Product3' */
                 HardenedData::TripleData<real_T>    Sum_i;                           /* '<S111>/Sum' */
                 HardenedData::TripleData<real_T>    Divide;                          /* '<S9>/Divide' */
                 HardenedData::TripleData<real_T>    Product_j;                       /* '<S112>/Product' */
                 HardenedData::TripleData<real_T>    UnaryMinus;                      /* '<S110>/Unary Minus' */
                 HardenedData::TripleData<real_T>    Divide1;                         /* '<S9>/Divide1' */
                 HardenedData::TripleData<real_T>    Product1_h;                      /* '<S112>/Product1' */
                 HardenedData::TripleData<real_T>    UnaryMinus1;                     /* '<S110>/Unary Minus1' */
                 HardenedData::TripleData<real_T>    Divide2;                         /* '<S9>/Divide2' */
                 HardenedData::TripleData<real_T>    Product2_d;                      /* '<S112>/Product2' */
                 HardenedData::TripleData<real_T>    UnaryMinus2;                     /* '<S110>/Unary Minus2' */
                 HardenedData::TripleData<real_T>    Divide3;                         /* '<S9>/Divide3' */
                 HardenedData::TripleData<real_T>    Product3_c;                      /* '<S112>/Product3' */
                 HardenedData::TripleData<real_T>    Sum_p;                           /* '<S112>/Sum' */
                 HardenedData::TripleData<real_T>    Product_b;                       /* '<S123>/Product' */
                 HardenedData::TripleData<real_T>    Product_p;                       /* '<S113>/Product' */
                 HardenedData::TripleData<real_T>    Product1_i;                      /* '<S113>/Product1' */
                 HardenedData::TripleData<real_T>    Product2_o;                      /* '<S113>/Product2' */
                 HardenedData::TripleData<real_T>    Product3_p;                      /* '<S113>/Product3' */
                 HardenedData::TripleData<real_T>    Sum_iw;                          /* '<S113>/Sum' */
                 HardenedData::TripleData<real_T>    Product1_in;                     /* '<S123>/Product1' */
                 HardenedData::TripleData<real_T>    Product_by;                      /* '<S114>/Product' */
                 HardenedData::TripleData<real_T>    Product1_f;                      /* '<S114>/Product1' */
                 HardenedData::TripleData<real_T>    Product2_m;                      /* '<S114>/Product2' */
                 HardenedData::TripleData<real_T>    Product3_pa;                     /* '<S114>/Product3' */
                 HardenedData::TripleData<real_T>    Sum_d;                           /* '<S114>/Sum' */
                 HardenedData::TripleData<real_T>    Product2_n;                      /* '<S123>/Product2' */
                 HardenedData::TripleData<real_T>    Product_pd;                      /* '<S115>/Product' */
                 HardenedData::TripleData<real_T>    Product1_d;                      /* '<S115>/Product1' */
                 HardenedData::TripleData<real_T>    Product2_nn;                     /* '<S115>/Product2' */
                 HardenedData::TripleData<real_T>    Product3_l;                      /* '<S115>/Product3' */
                 HardenedData::TripleData<real_T>    Sum_o;                           /* '<S115>/Sum' */
                 HardenedData::TripleData<real_T>    Product3_ph;                     /* '<S123>/Product3' */
                 HardenedData::TripleData<real_T>    Sum_g;                           /* '<S123>/Sum' */
                 HardenedData::TripleData<real_T>    sqrt_g;                          /* '<S122>/sqrt' */
                 HardenedData::TripleData<real_T>    Product_h;                       /* '<S117>/Product' */
                 HardenedData::TripleData<real_T>    Product1_j;                      /* '<S117>/Product1' */
                 HardenedData::TripleData<real_T>    Product2_dj;                     /* '<S117>/Product2' */
                 HardenedData::TripleData<real_T>    Product3_e;                      /* '<S117>/Product3' */
                 HardenedData::TripleData<real_T>    fcn1;                            /* '<S11>/fcn1' */
                 HardenedData::TripleData<real_T>    fcn2;                            /* '<S11>/fcn2' */
                 HardenedData::TripleData<real_T>    fcn3;                            /* '<S11>/fcn3' */
                 HardenedData::TripleData<real_T>    Merge_il;                        /* '<S118>/Merge' */
                 HardenedData::TripleData<real_T>    fcn4;                            /* '<S11>/fcn4' */
                 HardenedData::TripleData<real_T>    fcn5;                            /* '<S11>/fcn5' */
                 HardenedData::TripleData<real_T>    VectorConcatenate[3];            /* '<S116>/Vector Concatenate' */
                 HardenedData::TripleData<real_T>    DerivativeGain[3];               /* '<S85>/Derivative Gain' */
                 HardenedData::TripleData<real_T>    Tsamp[3];                        /* '<S88>/Tsamp' */
                 HardenedData::TripleData<real_T>    UD[3];                           /* '<S86>/UD' */
                 HardenedData::TripleData<real_T>    Diff[3];                         /* '<S86>/Diff' */
                 HardenedData::TripleData<real_T>    IntegralGain[3];                 /* '<S90>/Integral Gain' */
                 HardenedData::TripleData<real_T>    Integrator[3];                   /* '<S93>/Integrator' */
                 HardenedData::TripleData<real_T>    ProportionalGain[3];             /* '<S98>/Proportional Gain' */
                 HardenedData::TripleData<real_T>    Product_a;                       /* '<S58>/Product' */
                 HardenedData::TripleData<real_T>    Product1_k;                      /* '<S58>/Product1' */
                 HardenedData::TripleData<real_T>    Product2_e;                      /* '<S58>/Product2' */
                 HardenedData::TripleData<real_T>    Product3_g;                      /* '<S58>/Product3' */
                 HardenedData::TripleData<real_T>    Product4;                        /* '<S58>/Product4' */
                 HardenedData::TripleData<real_T>    Product5;                        /* '<S58>/Product5' */
```

```
HardenedData::TripleData<real_T>    detMatrix;                         /* '<S58>/Sum' */
HardenedData::TripleData<real_T>    Bias;                              /* '<S52>/Bias' */
HardenedData::TripleData<real_T>    Abs1;                              /* '<S52>/Abs1' */
HardenedData::TripleData<real_T>    MathFunction[9];                   /* '<S51>/Math Function' */
HardenedData::TripleData<real_T>    Product_c[9];                      /* '<S51>/Product' */
HardenedData::TripleData<real_T>    Bias1[9];                          /* '<S51>/Bias1' */
HardenedData::TripleData<real_T>    Abs2[9];                           /* '<S51>/Abs2' */
HardenedData::TripleData<real_T>    Add_m;                             /* '<S34>/Add' */
HardenedData::TripleData<real_T>    sqrt_m;                            /* '<S26>/sqrt' */
HardenedData::TripleData<real_T>    Switch[2];                         /* '<S33>/Switch' */
HardenedData::TripleData<real_T>    Product_k;                         /* '<S33>/Product' */
HardenedData::TripleData<real_T>    Add_o;                             /* '<S32>/Add' */
HardenedData::TripleData<real_T>    Add_ov;                            /* '<S30>/Add' */
HardenedData::TripleData<real_T>    Add_l;                             /* '<S31>/Add' */
HardenedData::TripleData<real_T>    Product_cy[3];                     /* '<S26>/Product' */
HardenedData::TripleData<real_T>    Add_f;                             /* '<S44>/Add' */
HardenedData::TripleData<real_T>    sqrt_l;                            /* '<S28>/sqrt' */
HardenedData::TripleData<real_T>    Add_e;                             /* '<S40>/Add' */
HardenedData::TripleData<real_T>    Add_l2;                            /* '<S41>/Add' */
HardenedData::TripleData<real_T>    Add_ek;                            /* '<S42>/Add' */
HardenedData::TripleData<real_T>    Switch_b[2];                       /* '<S43>/Switch' */
HardenedData::TripleData<real_T>    Product_d;                         /* '<S43>/Product' */
HardenedData::TripleData<real_T>    Product_i[3];                      /* '<S28>/Product' */
HardenedData::TripleData<real_T>    Add_g;                             /* '<S39>/Add' */
HardenedData::TripleData<real_T>    sqrt_mw;                           /* '<S27>/sqrt' */
HardenedData::TripleData<real_T>    Add_gw;                            /* '<S37>/Add' */
HardenedData::TripleData<real_T>    Add_fc;                            /* '<S36>/Add' */
HardenedData::TripleData<real_T>    Add_d;                             /* '<S35>/Add' */
HardenedData::TripleData<real_T>    Switch_f[2];                       /* '<S38>/Switch' */
HardenedData::TripleData<real_T>    Product_e;                         /* '<S38>/Product' */
HardenedData::TripleData<real_T>    Product_o[3];                      /* '<S27>/Product' */
HardenedData::TripleData<real_T>    Sum_gh;                            /* '<S23>/Sum' */
HardenedData::TripleData<real_T>    sqrt_a;                            /* '<S23>/sqrt' */
HardenedData::TripleData<real_T>    Gain1;                             /* '<S23>/Gain1' */
HardenedData::TripleData<real_T>    Add_k;                             /* '<S46>/Add' */
HardenedData::TripleData<real_T>    Add_el;                            /* '<S45>/Add' */
HardenedData::TripleData<real_T>    Add_a;                             /* '<S47>/Add' */
HardenedData::TripleData<real_T>    Merge_m;                           /* '<S129>/Merge' */
HardenedData::TripleData<real_T>    Product_n;                         /* '<S259>/Product' */
HardenedData::TripleData<real_T>    Product1_ii;                       /* '<S259>/Product1' */
HardenedData::TripleData<real_T>    Product2_c;                        /* '<S259>/Product2' */
HardenedData::TripleData<real_T>    Product3_d;                        /* '<S259>/Product3' */
HardenedData::TripleData<real_T>    Sum_j;                             /* '<S259>/Sum' */
HardenedData::TripleData<real_T>    sqrt_n;                            /* '<S258>/sqrt' */
HardenedData::TripleData<real_T>    Product2_oi;                       /* '<S254>/Product2' */
HardenedData::TripleData<real_T>    Product6;                          /* '<S255>/Product6' */
HardenedData::TripleData<real_T>    Product3_a;                        /* '<S254>/Product3' */
HardenedData::TripleData<real_T>    Product7;                          /* '<S255>/Product7' */
HardenedData::TripleData<real_T>    Sum3;                              /* '<S255>/Sum3' */
HardenedData::TripleData<real_T>    Gain2;                             /* '<S255>/Gain2' */
HardenedData::TripleData<real_T>    Product8;                          /* '<S255>/Product8' */
HardenedData::TripleData<real_T>    Product1_fv;                       /* '<S254>/Product1' */
HardenedData::TripleData<real_T>    Product_jb;                        /* '<S255>/Product' */
HardenedData::TripleData<real_T>    Product_cw;                        /* '<S254>/Product' */
HardenedData::TripleData<real_T>    Product1_b;                        /* '<S255>/Product1' */
HardenedData::TripleData<real_T>    Sum1;                              /* '<S255>/Sum1' */
HardenedData::TripleData<real_T>    Gain;                              /* '<S255>/Gain' */
HardenedData::TripleData<real_T>    Product4_p;                        /* '<S255>/Product4' */
HardenedData::TripleData<real_T>    Product2_ox;                       /* '<S255>/Product2' */
HardenedData::TripleData<real_T>    Product3_pv;                       /* '<S255>/Product3' */
HardenedData::TripleData<real_T>    Sum2;                              /* '<S255>/Sum2' */
HardenedData::TripleData<real_T>    Gain1_l;                           /* '<S255>/Gain1' */
HardenedData::TripleData<real_T>    Product5_g;                        /* '<S255>/Product5' */
HardenedData::TripleData<real_T>    Sum_k;                             /* '<S255>/Sum' */
HardenedData::TripleData<real_T>    Product_if;                        /* '<S256>/Product' */
HardenedData::TripleData<real_T>    Product1_e;                        /* '<S256>/Product1' */
HardenedData::TripleData<real_T>    Sum1_b;                            /* '<S256>/Sum1' */
HardenedData::TripleData<real_T>    Gain_a;                            /* '<S256>/Gain' */
HardenedData::TripleData<real_T>    Product4_k;                        /* '<S256>/Product4' */
HardenedData::TripleData<real_T>    Product6_d;                        /* '<S256>/Product6' */
HardenedData::TripleData<real_T>    Product7_e;                        /* '<S256>/Product7' */
HardenedData::TripleData<real_T>    Sum3_p;                            /* '<S256>/Sum3' */
HardenedData::TripleData<real_T>    Gain2_a;                           /* '<S256>/Gain2' */
HardenedData::TripleData<real_T>    Product8_c;                        /* '<S256>/Product8' */
HardenedData::TripleData<real_T>    Product2_l;                        /* '<S256>/Product2' */
HardenedData::TripleData<real_T>    Product3_c5;                       /* '<S256>/Product3' */
HardenedData::TripleData<real_T>    Sum2_k;                            /* '<S256>/Sum2' */
HardenedData::TripleData<real_T>    Gain1_d;                           /* '<S256>/Gain1' */
HardenedData::TripleData<real_T>    Product5_j;                        /* '<S256>/Product5' */
HardenedData::TripleData<real_T>    Sum_lm;                            /* '<S256>/Sum' */
HardenedData::TripleData<real_T>    Product_c3;                        /* '<S257>/Product' */
HardenedData::TripleData<real_T>    Product1_c;                        /* '<S257>/Product1' */
HardenedData::TripleData<real_T>    Sum1_bu;                           /* '<S257>/Sum1' */
HardenedData::TripleData<real_T>    Gain_k;                            /* '<S257>/Gain' */
HardenedData::TripleData<real_T>    Product4_m;                        /* '<S257>/Product4' */
HardenedData::TripleData<real_T>    Product2_k;                        /* '<S257>/Product2' */
HardenedData::TripleData<real_T>    Product3_f;                        /* '<S257>/Product3' */
HardenedData::TripleData<real_T>    Sum2_p;                            /* '<S257>/Sum2' */
HardenedData::TripleData<real_T>    Gain1_i;                           /* '<S257>/Gain1' */
HardenedData::TripleData<real_T>    Product5_d;                        /* '<S257>/Product5' */
HardenedData::TripleData<real_T>    Product6_b;                        /* '<S257>/Product6' */
HardenedData::TripleData<real_T>    Product7_h;                        /* '<S257>/Product7' */
HardenedData::TripleData<real_T>    Sum3_e;                            /* '<S257>/Sum3' */
HardenedData::TripleData<real_T>    Gain2_b;                           /* '<S257>/Gain2' */
HardenedData::TripleData<real_T>    Product8_g;                        /* '<S257>/Product8' */
HardenedData::TripleData<real_T>    Sum_n;                             /* '<S257>/Sum' */
HardenedData::TripleData<real_T>    TmpSignalConversionAtDotProduct[3];
HardenedData::TripleData<real_T>    Product_jz;                        /* '<S265>/Product' */
HardenedData::TripleData<real_T>    Product1_bp;                       /* '<S265>/Product1' */
HardenedData::TripleData<real_T>    Product2_b;                        /* '<S265>/Product2' */
HardenedData::TripleData<real_T>    Product3_dh;                       /* '<S265>/Product3' */
HardenedData::TripleData<real_T>    Sum_d5;                            /* '<S265>/Sum' */
HardenedData::TripleData<real_T>    sqrt_k;                            /* '<S264>/sqrt' */
HardenedData::TripleData<real_T>    Product2_lq;                       /* '<S260>/Product2' */
```

```
    HardenedData::TripleData<real_T>    Product6_l;                    /* '<S261>/Product6' */
    HardenedData::TripleData<real_T>    Product3_h;                    /* '<S260>/Product3' */
    HardenedData::TripleData<real_T>    Product7_d;                    /* '<S261>/Product7' */
    HardenedData::TripleData<real_T>    Sum3_j;                        /* '<S261>/Sum3' */
    HardenedData::TripleData<real_T>    Gain2_f;                       /* '<S261>/Gain2' */
    HardenedData::TripleData<real_T>    Product8_m;                    /* '<S261>/Product8' */
    HardenedData::TripleData<real_T>    Product1_b1;                   /* '<S260>/Product1' */
    HardenedData::TripleData<real_T>    Product_h1;                    /* '<S261>/Product' */
    HardenedData::TripleData<real_T>    Product_f;                     /* '<S260>/Product' */
    HardenedData::TripleData<real_T>    Product1_l;                    /* '<S261>/Product1' */
    HardenedData::TripleData<real_T>    Sum1_d;                        /* '<S261>/Sum1' */
    HardenedData::TripleData<real_T>    Gain_j;                        /* '<S261>/Gain' */
    HardenedData::TripleData<real_T>    Product4_o;                    /* '<S261>/Product4' */
    HardenedData::TripleData<real_T>    Product2_p;                    /* '<S261>/Product2' */
    HardenedData::TripleData<real_T>    Product3_lx;                   /* '<S261>/Product3' */
    HardenedData::TripleData<real_T>    Sum2_p3;                       /* '<S261>/Sum2' */
    HardenedData::TripleData<real_T>    Gain1_o;                       /* '<S261>/Gain1' */
    HardenedData::TripleData<real_T>    Product5_i;                    /* '<S261>/Product5' */
    HardenedData::TripleData<real_T>    Sum_kk;                        /* '<S261>/Sum' */
    HardenedData::TripleData<real_T>    Product_bg;                    /* '<S262>/Product' */
    HardenedData::TripleData<real_T>    Product1_l3;                   /* '<S262>/Product1' */
    HardenedData::TripleData<real_T>    Sum1_l;                        /* '<S262>/Sum1' */
    HardenedData::TripleData<real_T>    Gain_i;                        /* '<S262>/Gain' */
    HardenedData::TripleData<real_T>    Product4_pt;                   /* '<S262>/Product4' */
    HardenedData::TripleData<real_T>    Product6_p;                    /* '<S262>/Product6' */
    HardenedData::TripleData<real_T>    Product7_dj;                   /* '<S262>/Product7' */
    HardenedData::TripleData<real_T>    Sum3_o;                        /* '<S262>/Sum3' */
    HardenedData::TripleData<real_T>    Gain2_j;                       /* '<S262>/Gain2' */
    HardenedData::TripleData<real_T>    Product8_e;                    /* '<S262>/Product8' */
    HardenedData::TripleData<real_T>    Product2_mf;                   /* '<S262>/Product2' */
    HardenedData::TripleData<real_T>    Product3_k;                    /* '<S262>/Product3' */
    HardenedData::TripleData<real_T>    Sum2_a;                        /* '<S262>/Sum2' */
    HardenedData::TripleData<real_T>    Gain1_oi;                      /* '<S262>/Gain1' */
    HardenedData::TripleData<real_T>    Product5_b;                    /* '<S262>/Product5' */
    HardenedData::TripleData<real_T>    Sum_ln;                        /* '<S262>/Sum' */
    HardenedData::TripleData<real_T>    Product_l;                     /* '<S263>/Product' */
    HardenedData::TripleData<real_T>    Product1_id;                   /* '<S263>/Product1' */
    HardenedData::TripleData<real_T>    Sum1_d1;                       /* '<S263>/Sum1' */
    HardenedData::TripleData<real_T>    Gain_o;                        /* '<S263>/Gain' */
    HardenedData::TripleData<real_T>    Product4_d;                    /* '<S263>/Product4' */
    HardenedData::TripleData<real_T>    Product2_f;                    /* '<S263>/Product2' */
    HardenedData::TripleData<real_T>    Product3_gx;                   /* '<S263>/Product3' */
    HardenedData::TripleData<real_T>    Sum2_l;                        /* '<S263>/Sum2' */
    HardenedData::TripleData<real_T>    Gain1_g;                       /* '<S263>/Gain1' */
    HardenedData::TripleData<real_T>    Product5_n;                    /* '<S263>/Product5' */
    HardenedData::TripleData<real_T>    Product6_j;                    /* '<S263>/Product6' */
    HardenedData::TripleData<real_T>    Product7_g;                    /* '<S263>/Product7' */
    HardenedData::TripleData<real_T>    Sum3_b;                        /* '<S263>/Sum3' */
    HardenedData::TripleData<real_T>    Gain2_o;                       /* '<S263>/Gain2' */
    HardenedData::TripleData<real_T>    Product8_a;                    /* '<S263>/Product8' */
    HardenedData::TripleData<real_T>    Sum_kb;                        /* '<S263>/Sum' */
    HardenedData::TripleData<real_T>    TmpSignalConversionAtDotProdu_d[3];
    HardenedData::TripleData<real_T>    DotProduct3;                   /* '<S130>/Dot Product3' */
    HardenedData::TripleData<real_T>    DotProduct1;                   /* '<S130>/Dot Product1' */
    HardenedData::TripleData<real_T>    DotProduct2;                   /* '<S130>/Dot Product2' */
    HardenedData::TripleData<real_T>    Divide1_k;                     /* '<S130>/Divide1' */
    HardenedData::TripleData<real_T>    Sqrt3;                         /* '<S130>/Sqrt3' */
    HardenedData::TripleData<real_T>    Divide_o;                      /* '<S130>/Divide' */
    HardenedData::TripleData<real_T>    Bias_k;                        /* '<S130>/Bias' */
    HardenedData::TripleData<real_T>    Abs;                           /* '<S130>/Abs' */
    HardenedData::TripleData<real_T>    Bias1_j;                       /* '<S130>/Bias1' */
    HardenedData::TripleData<real_T>    Abs1_d;                        /* '<S130>/Abs1' */
    HardenedData::TripleData<real_T>    Merge_ip;                      /* '<S130>/Merge' */
    HardenedData::TripleData<real_T>    DotProduct3_o;                 /* '<S136>/Dot Product3' */
    HardenedData::TripleData<real_T>    DotProduct2_c;                 /* '<S136>/Dot Product2' */
    HardenedData::TripleData<real_T>    Divide1_d;                     /* '<S136>/Divide1' */
    HardenedData::TripleData<real_T>    Sqrt3_j;                       /* '<S136>/Sqrt3' */
    HardenedData::TripleData<real_T>    Add2;                          /* '<S136>/Add2' */
    HardenedData::TripleData<real_T>    Product_i2;                    /* '<S176>/Product' */
    HardenedData::TripleData<real_T>    DotProduct3_p;                 /* '<S163>/Dot Product3' */
    HardenedData::TripleData<real_T>    DotProduct2_f;                 /* '<S163>/Dot Product2' */
    HardenedData::TripleData<real_T>    Divide1_b;                     /* '<S163>/Divide1' */
    HardenedData::TripleData<real_T>    Sqrt3_l;                       /* '<S163>/Sqrt3' */
    HardenedData::TripleData<real_T>    Divide_k;                      /* '<S163>/Divide' */
    HardenedData::TripleData<real_T>    Bias_b;                        /* '<S163>/Bias' */
    HardenedData::TripleData<real_T>    Abs_o;                         /* '<S163>/Abs' */
    HardenedData::TripleData<real_T>    Bias1_m;                       /* '<S163>/Bias1' */
    HardenedData::TripleData<real_T>    Abs1_l;                        /* '<S163>/Abs1' */
    HardenedData::TripleData<real_T>    Merge_d;                       /* '<S163>/Merge' */
    HardenedData::TripleData<real_T>    is180degRot[3];                /* '<S136>/is 180deg Rot' */
    HardenedData::TripleData<real_T>    Product1_g;                    /* '<S176>/Product1' */
    HardenedData::TripleData<real_T>    Product2_g;                    /* '<S176>/Product2' */
    HardenedData::TripleData<real_T>    Product3_aj;                   /* '<S176>/Product3' */
    HardenedData::TripleData<real_T>    Sum_c;                         /* '<S176>/Sum' */
    HardenedData::TripleData<real_T>    sqrt_h;                        /* '<S175>/sqrt' */
    HardenedData::TripleData<real_T>    Product_fv;                    /* '<S166>/Product' */
    HardenedData::TripleData<real_T>    Product2_a;                    /* '<S166>/Product2' */
    HardenedData::TripleData<real_T>    Product_l5;                    /* '<S155>/Product' */
    HardenedData::TripleData<real_T>    Product1_n;                    /* '<S166>/Product1' */
    HardenedData::TripleData<real_T>    Product1_lg;                   /* '<S155>/Product1' */
    HardenedData::TripleData<real_T>    Product2_fh;                   /* '<S155>/Product2' */
    HardenedData::TripleData<real_T>    Product3_pe;                   /* '<S166>/Product3' */
    HardenedData::TripleData<real_T>    Product3_l0;                   /* '<S155>/Product3' */
    HardenedData::TripleData<real_T>    Sum_n3;                        /* '<S155>/Sum' */
    HardenedData::TripleData<real_T>    sqrt_mv;                       /* '<S154>/sqrt' */
    HardenedData::TripleData<real_T>    Product2_ke;                   /* '<S150>/Product2' */
    HardenedData::TripleData<real_T>    Product6_e;                    /* '<S151>/Product6' */
    HardenedData::TripleData<real_T>    Product3_cc;                   /* '<S150>/Product3' */
    HardenedData::TripleData<real_T>    Product7_a;                    /* '<S151>/Product7' */
    HardenedData::TripleData<real_T>    Sum3_b5;                       /* '<S151>/Sum3' */
    HardenedData::TripleData<real_T>    Gain2_p;                       /* '<S151>/Gain2' */
    HardenedData::TripleData<real_T>    Product8_k;                    /* '<S151>/Product8' */
    HardenedData::TripleData<real_T>    Product1_l0;                   /* '<S150>/Product1' */
    HardenedData::TripleData<real_T>    Product_da;                    /* '<S151>/Product' */
```

```
    HardenedData::TripleData<real_T>  Product_m;                          /* '<S150>/Product' */
    HardenedData::TripleData<real_T>  Product1_dj;                        /* '<S151>/Product1' */
    HardenedData::TripleData<real_T>  Sum1_n;                             /* '<S151>/Sum1' */
    HardenedData::TripleData<real_T>  Gain_l;                             /* '<S151>/Gain' */
    HardenedData::TripleData<real_T>  Product4_c;                         /* '<S151>/Product4' */
    HardenedData::TripleData<real_T>  Product2_gk;                        /* '<S151>/Product2' */
    HardenedData::TripleData<real_T>  Product3_i;                         /* '<S151>/Product3' */
    HardenedData::TripleData<real_T>  Sum2_g;                             /* '<S151>/Sum2' */
    HardenedData::TripleData<real_T>  Gain1_e;                            /* '<S151>/Gain1' */
    HardenedData::TripleData<real_T>  Product5_o;                         /* '<S151>/Product5' */
    HardenedData::TripleData<real_T>  Sum_oz;                             /* '<S151>/Sum' */
    HardenedData::TripleData<real_T>  Product_g;                          /* '<S152>/Product' */
    HardenedData::TripleData<real_T>  Product1_ew;                        /* '<S152>/Product1' */
    HardenedData::TripleData<real_T>  Sum1_m;                             /* '<S152>/Sum1' */
    HardenedData::TripleData<real_T>  Gain_b;                             /* '<S152>/Gain' */
    HardenedData::TripleData<real_T>  Product4_a;                         /* '<S152>/Product4' */
    HardenedData::TripleData<real_T>  Product6_o;                         /* '<S152>/Product6' */
    HardenedData::TripleData<real_T>  Product7_k;                         /* '<S152>/Product7' */
    HardenedData::TripleData<real_T>  Sum3_bi;                            /* '<S152>/Sum3' */
    HardenedData::TripleData<real_T>  Gain2_l;                            /* '<S152>/Gain2' */
    HardenedData::TripleData<real_T>  Product8_me;                        /* '<S152>/Product8' */
    HardenedData::TripleData<real_T>  Product2_fhq;                       /* '<S152>/Product2' */
    HardenedData::TripleData<real_T>  Product3_n;                         /* '<S152>/Product3' */
    HardenedData::TripleData<real_T>  Sum2_la;                            /* '<S152>/Sum2' */
    HardenedData::TripleData<real_T>  Gain1_ox;                           /* '<S152>/Gain1' */
    HardenedData::TripleData<real_T>  Product5_l;                         /* '<S152>/Product5' */
    HardenedData::TripleData<real_T>  Sum_oa;                             /* '<S152>/Sum' */
    HardenedData::TripleData<real_T>  Product_ew;                         /* '<S153>/Product' */
    HardenedData::TripleData<real_T>  Product1_m;                         /* '<S153>/Product1' */
    HardenedData::TripleData<real_T>  Sum1_h;                             /* '<S153>/Sum1' */
    HardenedData::TripleData<real_T>  Gain_kp;                            /* '<S153>/Gain' */
    HardenedData::TripleData<real_T>  Product4_g;                         /* '<S153>/Product4' */
    HardenedData::TripleData<real_T>  Product2_gh;                        /* '<S153>/Product2' */
    HardenedData::TripleData<real_T>  Product3_k4;                        /* '<S153>/Product3' */
    HardenedData::TripleData<real_T>  Sum2_i;                             /* '<S153>/Sum2' */
    HardenedData::TripleData<real_T>  Gain1_f;                            /* '<S153>/Gain1' */
    HardenedData::TripleData<real_T>  Product5_ij;                        /* '<S153>/Product5' */
    HardenedData::TripleData<real_T>  Product6_c;                         /* '<S153>/Product6' */
    HardenedData::TripleData<real_T>  Product7_a4;                        /* '<S153>/Product7' */
    HardenedData::TripleData<real_T>  Sum3_d;                             /* '<S153>/Sum3' */
    HardenedData::TripleData<real_T>  Gain2_n;                            /* '<S153>/Gain2' */
    HardenedData::TripleData<real_T>  Product8_f;                         /* '<S153>/Product8' */
    HardenedData::TripleData<real_T>  Sum_nx;                             /* '<S153>/Sum' */
    HardenedData::TripleData<real_T>  TmpSignalConversionAtDotProdu_k[3];
    HardenedData::TripleData<real_T>  jxk_b;                              /* '<S200>/j x k' */
    HardenedData::TripleData<real_T>  kxi_b;                              /* '<S200>/k x i' */
    HardenedData::TripleData<real_T>  ixj_h;                              /* '<S200>/i x j' */
    HardenedData::TripleData<real_T>  kxj_l;                              /* '<S201>/k x j' */
    HardenedData::TripleData<real_T>  ixk_e;                              /* '<S201>/i x k' */
    HardenedData::TripleData<real_T>  jxi_n;                              /* '<S201>/j x i' */
    HardenedData::TripleData<real_T>  Sum_m[3];                           /* '<S198>/Sum' */
    HardenedData::TripleData<real_T>  jxk_e;                              /* '<S202>/j x k' */
    HardenedData::TripleData<real_T>  kxi_es;                             /* '<S202>/k x i' */
    HardenedData::TripleData<real_T>  ixj_n;                              /* '<S202>/i x j' */
    HardenedData::TripleData<real_T>  kxj_j;                              /* '<S203>/k x j' */
    HardenedData::TripleData<real_T>  ixk_j;                              /* '<S203>/i x k' */
    HardenedData::TripleData<real_T>  jxi_f;                              /* '<S203>/j x i' */
    HardenedData::TripleData<real_T>  Sum_pb[3];                          /* '<S199>/Sum' */
    HardenedData::TripleData<real_T>  Product_dj;                         /* '<S161>/Product' */
    HardenedData::TripleData<real_T>  Product1_ig;                        /* '<S161>/Product1' */
    HardenedData::TripleData<real_T>  Product2_ad;                        /* '<S161>/Product2' */
    HardenedData::TripleData<real_T>  Product3_hf;                        /* '<S161>/Product3' */
    HardenedData::TripleData<real_T>  Sum_gt;                             /* '<S161>/Sum' */
    HardenedData::TripleData<real_T>  sqrt_i;                             /* '<S160>/sqrt' */
    HardenedData::TripleData<real_T>  Product2_nx;                        /* '<S156>/Product2' */
    HardenedData::TripleData<real_T>  Product6_a;                         /* '<S157>/Product6' */
    HardenedData::TripleData<real_T>  Product3_ce;                        /* '<S156>/Product3' */
    HardenedData::TripleData<real_T>  Product7_ha;                        /* '<S157>/Product7' */
    HardenedData::TripleData<real_T>  Sum3_l;                             /* '<S157>/Sum3' */
    HardenedData::TripleData<real_T>  Gain2_om;                           /* '<S157>/Gain2' */
    HardenedData::TripleData<real_T>  Product8_p;                         /* '<S157>/Product8' */
    HardenedData::TripleData<real_T>  Product1_a;                         /* '<S156>/Product1' */
    HardenedData::TripleData<real_T>  Product_mi;                         /* '<S157>/Product' */
    HardenedData::TripleData<real_T>  Product_pb;                         /* '<S156>/Product' */
    HardenedData::TripleData<real_T>  Product1_lr;                        /* '<S157>/Product1' */
    HardenedData::TripleData<real_T>  Sum1_a;                             /* '<S157>/Sum1' */
    HardenedData::TripleData<real_T>  Gain_p;                             /* '<S157>/Gain' */
    HardenedData::TripleData<real_T>  Product4_i;                         /* '<S157>/Product4' */
    HardenedData::TripleData<real_T>  Product2_nz;                        /* '<S157>/Product2' */
    HardenedData::TripleData<real_T>  Product3_lu;                        /* '<S157>/Product3' */
    HardenedData::TripleData<real_T>  Sum2_m;                             /* '<S157>/Sum2' */
    HardenedData::TripleData<real_T>  Gain1_c;                            /* '<S157>/Gain1' */
    HardenedData::TripleData<real_T>  Product5_jx;                        /* '<S157>/Product5' */
    HardenedData::TripleData<real_T>  Sum_g5;                             /* '<S157>/Sum' */
    HardenedData::TripleData<real_T>  Product_pe;                         /* '<S158>/Product' */
    HardenedData::TripleData<real_T>  Product1_mu;                        /* '<S158>/Product1' */
    HardenedData::TripleData<real_T>  Sum1_k;                             /* '<S158>/Sum1' */
    HardenedData::TripleData<real_T>  Gain_ay;                            /* '<S158>/Gain' */
    HardenedData::TripleData<real_T>  Product4_ok;                        /* '<S158>/Product4' */
    HardenedData::TripleData<real_T>  Product6_ld;                        /* '<S158>/Product6' */
    HardenedData::TripleData<real_T>  Product7_ei;                        /* '<S158>/Product7' */
    HardenedData::TripleData<real_T>  Sum3_h;                             /* '<S158>/Sum3' */
    HardenedData::TripleData<real_T>  Gain2_l2;                           /* '<S158>/Gain2' */
    HardenedData::TripleData<real_T>  Product8_d;                         /* '<S158>/Product8' */
    HardenedData::TripleData<real_T>  Product2_ea;                        /* '<S158>/Product2' */
    HardenedData::TripleData<real_T>  Product3_ll;                        /* '<S158>/Product3' */
    HardenedData::TripleData<real_T>  Sum2_mr;                            /* '<S158>/Sum2' */
    HardenedData::TripleData<real_T>  Gain1_n;                            /* '<S158>/Gain1' */
    HardenedData::TripleData<real_T>  Product5_a;                         /* '<S158>/Product5' */
    HardenedData::TripleData<real_T>  Sum_gz;                             /* '<S158>/Sum' */
    HardenedData::TripleData<real_T>  Product_am;                         /* '<S159>/Product' */
    HardenedData::TripleData<real_T>  Product1_bt;                        /* '<S159>/Product1' */
    HardenedData::TripleData<real_T>  Sum1_hk;                            /* '<S159>/Sum1' */
    HardenedData::TripleData<real_T>  Gain_g;                             /* '<S159>/Gain' */
```

```
    HardenedData::TripleData<real_T>    Product4_o5;                        /* '<S159>/Product4' */
    HardenedData::TripleData<real_T>    Product2_cc;                        /* '<S159>/Product2' */
    HardenedData::TripleData<real_T>    Product3_cd;                        /* '<S159>/Product3' */
    HardenedData::TripleData<real_T>    Sum2_o;                             /* '<S159>/Sum2' */
    HardenedData::TripleData<real_T>    Gain1_ni;                           /* '<S159>/Gain1' */
    HardenedData::TripleData<real_T>    Product5_n2;                        /* '<S159>/Product5' */
    HardenedData::TripleData<real_T>    Product6_d5;                        /* '<S159>/Product6' */
    HardenedData::TripleData<real_T>    Product7_em;                        /* '<S159>/Product7' */
    HardenedData::TripleData<real_T>    Sum3_jh;                            /* '<S159>/Sum3' */
    HardenedData::TripleData<real_T>    Gain2_ja;                           /* '<S159>/Gain2' */
    HardenedData::TripleData<real_T>    Product8_j;                         /* '<S159>/Product8' */
    HardenedData::TripleData<real_T>    Sum_pw;                             /* '<S159>/Sum' */
    HardenedData::TripleData<real_T>    TmpSignalConversionAtDotProdu_j[3];
    HardenedData::TripleData<real_T>    DotProduct3_ph;                     /* '<S178>/Dot Product3' */
    HardenedData::TripleData<real_T>    DotProduct1_o;                      /* '<S178>/Dot Product1' */
    HardenedData::TripleData<real_T>    DotProduct2_g;                      /* '<S178>/Dot Product2' */
    HardenedData::TripleData<real_T>    Divide1_p;                          /* '<S178>/Divide1' */
    HardenedData::TripleData<real_T>    Sqrt3_m;                            /* '<S178>/Sqrt3' */
    HardenedData::TripleData<real_T>    Divide_c;                           /* '<S178>/Divide' */
    HardenedData::TripleData<real_T>    Bias_h;                             /* '<S178>/Bias' */
    HardenedData::TripleData<real_T>    Abs_m;                              /* '<S178>/Abs' */
    HardenedData::TripleData<real_T>    Bias1_e;                            /* '<S178>/Bias1' */
    HardenedData::TripleData<real_T>    Abs1_i;                             /* '<S178>/Abs1' */
    HardenedData::TripleData<real_T>    Merge_l;                            /* '<S178>/Merge' */
    HardenedData::TripleData<real_T>    Switch_c[4];                        /* '<S124>/Switch' */
    HardenedData::TripleData<real_T>    Product_h0;                         /* '<S273>/Product' */
    HardenedData::TripleData<real_T>    Product1_fm;                        /* '<S273>/Product1' */
    HardenedData::TripleData<real_T>    Product2_c2;                        /* '<S273>/Product2' */
    HardenedData::TripleData<real_T>    Product3_pq;                        /* '<S273>/Product3' */
    HardenedData::TripleData<real_T>    Sum_f;                              /* '<S273>/Sum' */
    HardenedData::TripleData<real_T>    sqrt_mo;                            /* '<S272>/sqrt' */
    HardenedData::TripleData<real_T>    Product_jc;                         /* '<S267>/Product' */
    HardenedData::TripleData<real_T>    Product1_o;                         /* '<S267>/Product1' */
    HardenedData::TripleData<real_T>    Product2_j;                         /* '<S267>/Product2' */
    HardenedData::TripleData<real_T>    Product3_lz;                        /* '<S267>/Product3' */
    HardenedData::TripleData<real_T>    fcn1_i;                             /* '<S128>/fcn1' */
    HardenedData::TripleData<real_T>    fcn2_f;                             /* '<S128>/fcn2' */
    HardenedData::TripleData<real_T>    fcn3_o;                             /* '<S128>/fcn3' */
    HardenedData::TripleData<real_T>    Merge_k;                            /* '<S268>/Merge' */
    HardenedData::TripleData<real_T>    fcn4_n;                             /* '<S128>/fcn4' */
    HardenedData::TripleData<real_T>    fcn5_g;                             /* '<S128>/fcn5' */
    HardenedData::TripleData<real_T>    VectorConcatenate_a[3];             /* '<S266>/Vector Concatenate' */
    HardenedData::TripleData<real_T>    DerivativeGain_k[3];                /* '<S229>/Derivative Gain' */
    HardenedData::TripleData<real_T>    Tsamp_f[3];                         /* '<S232>/Tsamp' */
    HardenedData::TripleData<real_T>    UD_l[3];                            /* '<S230>/UD' */
    HardenedData::TripleData<real_T>    Diff_e[3];                          /* '<S230>/Diff' */
    HardenedData::TripleData<real_T>    IntegralGain_p[3];                  /* '<S234>/Integral Gain' */
    HardenedData::TripleData<real_T>    Integrator_k[3];                    /* '<S237>/Integrator' */
    HardenedData::TripleData<real_T>    ProportionalGain_h[3];              /* '<S242>/Proportional Gain' */
    HardenedData::TripleData<real_T>    is180degRot_l[3];                   /* '<S137>/is 180deg Rot' */
    HardenedData::TripleData<real_T>    Product3_dz;                        /* '<S191>/Product3' */
    HardenedData::TripleData<real_T>    Product2_eq;                        /* '<S191>/Product2' */
    HardenedData::TripleData<real_T>    Product1_aa;                        /* '<S191>/Product1' */
    HardenedData::TripleData<real_T>    DotProduct2_n;                      /* '<S137>/Dot Product2' */
    HardenedData::TripleData<real_T>    DotProduct1_h;                      /* '<S137>/Dot Product1' */
    HardenedData::TripleData<real_T>    Divide1_h;                          /* '<S137>/Divide1' */
    HardenedData::TripleData<real_T>    Sqrt3_mq;                           /* '<S137>/Sqrt3' */
    HardenedData::TripleData<real_T>    DotProduct3_e;                      /* '<S137>/Dot Product3' */
    HardenedData::TripleData<real_T>    Add2_k;                             /* '<S137>/Add2' */
    HardenedData::TripleData<real_T>    Product_er;                         /* '<S191>/Product' */
    HardenedData::TripleData<real_T>    Sum_e;                              /* '<S191>/Sum' */
    HardenedData::TripleData<real_T>    sqrt_j;                             /* '<S190>/sqrt' */
    HardenedData::TripleData<real_T>    Product3_lzz;                       /* '<S181>/Product3' */
    HardenedData::TripleData<real_T>    Product3_ay;                        /* '<S146>/Product3' */
    HardenedData::TripleData<real_T>    Product2_ep;                        /* '<S181>/Product2' */
    HardenedData::TripleData<real_T>    Product2_bc;                        /* '<S146>/Product2' */
    HardenedData::TripleData<real_T>    Product1_ge;                        /* '<S181>/Product1' */
    HardenedData::TripleData<real_T>    Product1_fh;                        /* '<S146>/Product1' */
    HardenedData::TripleData<real_T>    Product_ey;                         /* '<S181>/Product' */
    HardenedData::TripleData<real_T>    Product_d0;                         /* '<S146>/Product' */
    HardenedData::TripleData<real_T>    Sum_ki;                             /* '<S146>/Sum' */
    HardenedData::TripleData<real_T>    Product3_aw;                        /* '<S147>/Product3' */
    HardenedData::TripleData<real_T>    Product2_pu;                        /* '<S147>/Product2' */
    HardenedData::TripleData<real_T>    Product1_b0;                        /* '<S147>/Product1' */
    HardenedData::TripleData<real_T>    Product_cq;                         /* '<S147>/Product' */
    HardenedData::TripleData<real_T>    Sum_oq;                             /* '<S147>/Sum' */
    HardenedData::TripleData<real_T>    Product3_h5;                        /* '<S148>/Product3' */
    HardenedData::TripleData<real_T>    Product2_i;                         /* '<S148>/Product2' */
    HardenedData::TripleData<real_T>    Product1_ai;                        /* '<S148>/Product1' */
    HardenedData::TripleData<real_T>    Product_cf;                         /* '<S148>/Product' */
    HardenedData::TripleData<real_T>    Sum_kc;                             /* '<S148>/Sum' */
    HardenedData::TripleData<real_T>    Product3_c1;                        /* '<S149>/Product3' */
    HardenedData::TripleData<real_T>    Product2_no;                        /* '<S149>/Product2' */
    HardenedData::TripleData<real_T>    Product1_d1;                        /* '<S149>/Product1' */
    HardenedData::TripleData<real_T>    Product_gr;                         /* '<S149>/Product' */
    HardenedData::TripleData<real_T>    Sum_n1;                             /* '<S149>/Sum' */
    HardenedData::TripleData<real_T>    Switch3[3];                         /* '<S180>/Switch3' */
    HardenedData::TripleData<real_T>    jxi_m;                              /* '<S189>/j x i' */
    HardenedData::TripleData<real_T>    ixk_k;                              /* '<S189>/i x k' */
    HardenedData::TripleData<real_T>    kxj_f;                              /* '<S189>/k x j' */
    HardenedData::TripleData<real_T>    ixj_k;                              /* '<S188>/i x j' */
    HardenedData::TripleData<real_T>    kxi_bj;                             /* '<S188>/k x i' */
    HardenedData::TripleData<real_T>    jxk_l;                              /* '<S188>/j x k' */
    HardenedData::TripleData<real_T>    Sum_pd[3];                          /* '<S187>/Sum' */
    HardenedData::TripleData<real_T>    Gain2_c;                            /* '<S180>/Gain2' */
    HardenedData::TripleData<real_T>    Gain_o2;                            /* '<S180>/Gain' */
    HardenedData::TripleData<real_T>    jxi_no;                             /* '<S183>/j x i' */
    HardenedData::TripleData<real_T>    ixk_b;                              /* '<S183>/i x k' */
    HardenedData::TripleData<real_T>    kxj_n;                              /* '<S183>/k x j' */
    HardenedData::TripleData<real_T>    ixj_l;                              /* '<S182>/i x j' */
    HardenedData::TripleData<real_T>    kxi_o;                              /* '<S182>/k x i' */
    HardenedData::TripleData<real_T>    jxk_d;                              /* '<S182>/j x k' */
    HardenedData::TripleData<real_T>    Sum_b[3];                           /* '<S177>/Sum' */
    HardenedData::TripleData<real_T>    jxi_k;                              /* '<S168>/j x i' */
```

```
                        HardenedData::TripleData<real_T>     ixk_a;                         /* '<S168>/i x k' */
                        HardenedData::TripleData<real_T>     kxj_c;                         /* '<S168>/k x j' */
                        HardenedData::TripleData<real_T>     ixj_o;                         /* '<S167>/i x j' */
                        HardenedData::TripleData<real_T>     kxi_p;                         /* '<S167>/k x i' */
                        HardenedData::TripleData<real_T>     jxk_er;                        /* '<S167>/j x k' */
                        HardenedData::TripleData<real_T>     Sum_ff[3];                     /* '<S162>/Sum' */
                HardenedData::TripleData<real_T>     Dipole_x[3];                   /* '<S2>/Magnetroquer control' */
                HardenedData::TripleData<real_T>     Dipole_y[3];                   /* '<S2>/Magnetroquer control' */
                HardenedData::TripleData<real_T>     Dipole_z[3];                   /* '<S2>/Magnetroquer control' */
                HardenedData::TripleData<real_T>     bdot[3];                       /* '<S2>/Derivative Function' */
#else
                HardenedData::SingleData<real_T>     norm_b;
                HardenedData::SingleData<real_T>     Merge[3];                      /* '<S1>/Merge' */
                                HardenedData::SingleData<real_T>     jxk;                           /* '<S20>/j x k' */
                                HardenedData::SingleData<real_T>     kxi;                           /* '<S20>/k x i' */
                                HardenedData::SingleData<real_T>     ixj;                           /* '<S20>/i x j' */
                                HardenedData::SingleData<real_T>     kxj;                           /* '<S21>/k x j' */
                                HardenedData::SingleData<real_T>     ixk;                           /* '<S21>/i x k' */
                                HardenedData::SingleData<real_T>     jxi;                           /* '<S21>/j x i' */
                                HardenedData::SingleData<real_T>     Sum[3];                        /* '<S13>/Sum' */
                                HardenedData::SingleData<real_T>     zr[3];                         /* '<S7>/z = -r' */
                                HardenedData::SingleData<real_T>     ixj_f;                         /* '<S18>/i x j' */
                                HardenedData::SingleData<real_T>     jxk_o;                         /* '<S18>/j x k' */
                                HardenedData::SingleData<real_T>     kxi_e;                         /* '<S18>/k x i' */
                                HardenedData::SingleData<real_T>     ixk_g;                         /* '<S19>/i x k' */
                                HardenedData::SingleData<real_T>     jxi_l;                         /* '<S19>/j x i' */
                                HardenedData::SingleData<real_T>     kxj_a;                         /* '<S19>/k x j' */
                                HardenedData::SingleData<real_T>     Sum_l[3];                      /* '<S12>/Sum' */
                                HardenedData::SingleData<real_T>     DotProduct;                    /* '<S15>/Dot Product' */
                                HardenedData::SingleData<real_T>     SumofElements;                 /* '<S15>/Sum of Elements' */
                                HardenedData::SingleData<real_T>     MathFunction1;                 /* '<S15>/Math Function1' */
                                HardenedData::SingleData<real_T>     DotProduct_p;                  /* '<S16>/Dot Product' */
                                HardenedData::SingleData<real_T>     SumofElements_c;               /* '<S16>/Sum of Elements' */
                                HardenedData::SingleData<real_T>     MathFunction1_b;               /* '<S16>/Math Function1' */
                                HardenedData::SingleData<real_T>     DotProduct_b;                  /* '<S17>/Dot Product' */
                                HardenedData::SingleData<real_T>     SumofElements_l;               /* '<S17>/Sum of Elements' */
                                HardenedData::SingleData<real_T>     MathFunction1_m;               /* '<S17>/Math Function1' */
                                HardenedData::SingleData<real_T>     ORFtoECEF[9];                  /* '<S7>/ORF to ECEF' */
                                HardenedData::SingleData<real_T>     ECEFtoORF[9];                  /* '<S7>/ECEF to ORF' */
                                HardenedData::SingleData<real_T>     Add;                           /* '<S25>/Add' */
                                HardenedData::SingleData<real_T>     Merge_i[4];                    /* '<S14>/Merge' */
                                HardenedData::SingleData<real_T>     Product;                       /* '<S111>/Product' */
                                HardenedData::SingleData<real_T>     Product1;                      /* '<S111>/Product1' */
                                HardenedData::SingleData<real_T>     Product2;                      /* '<S111>/Product2' */
                                HardenedData::SingleData<real_T>     Product3;                      /* '<S111>/Product3' */
                                HardenedData::SingleData<real_T>     Sum_i;                         /* '<S111>/Sum' */
                                HardenedData::SingleData<real_T>     Divide;                        /* '<S9>/Divide' */
                                HardenedData::SingleData<real_T>     Product_j;                     /* '<S112>/Product' */
                                HardenedData::SingleData<real_T>     UnaryMinus;                    /* '<S110>/Unary Minus' */
                                HardenedData::SingleData<real_T>     Divide1;                       /* '<S9>/Divide1' */
                                HardenedData::SingleData<real_T>     Product1_h;                    /* '<S112>/Product1' */
                                HardenedData::SingleData<real_T>     UnaryMinus1;                   /* '<S110>/Unary Minus1' */
                                HardenedData::SingleData<real_T>     Divide2;                       /* '<S9>/Divide2' */
                                HardenedData::SingleData<real_T>     Product2_d;                    /* '<S112>/Product2' */
                                HardenedData::SingleData<real_T>     UnaryMinus2;                   /* '<S110>/Unary Minus2' */
                                HardenedData::SingleData<real_T>     Divide3;                       /* '<S9>/Divide3' */
                                HardenedData::SingleData<real_T>     Product3_c;                    /* '<S112>/Product3' */
                                HardenedData::SingleData<real_T>     Sum_p;                         /* '<S112>/Sum' */
                                HardenedData::SingleData<real_T>     Product_b;                     /* '<S123>/Product' */
                                HardenedData::SingleData<real_T>     Product_p;                     /* '<S113>/Product' */
                                HardenedData::SingleData<real_T>     Product1_i;                    /* '<S113>/Product1' */
                                HardenedData::SingleData<real_T>     Product2_o;                    /* '<S113>/Product2' */
                                HardenedData::SingleData<real_T>     Product3_p;                    /* '<S113>/Product3' */
                                HardenedData::SingleData<real_T>     Sum_iw;                        /* '<S113>/Sum' */
                                HardenedData::SingleData<real_T>     Product1_in;                   /* '<S123>/Product1' */
                                HardenedData::SingleData<real_T>     Product_by;                    /* '<S114>/Product' */
                                HardenedData::SingleData<real_T>     Product1_f;                    /* '<S114>/Product1' */
                                HardenedData::SingleData<real_T>     Product2_m;                    /* '<S114>/Product2' */
                                HardenedData::SingleData<real_T>     Product3_pa;                   /* '<S114>/Product3' */
                                HardenedData::SingleData<real_T>     Sum_d;                         /* '<S114>/Sum' */
                                HardenedData::SingleData<real_T>     Product2_n;                    /* '<S123>/Product2' */
                                HardenedData::SingleData<real_T>     Product_pd;                    /* '<S115>/Product' */
                                HardenedData::SingleData<real_T>     Product1_d;                    /* '<S115>/Product1' */
                                HardenedData::SingleData<real_T>     Product2_nn;                   /* '<S115>/Product2' */
                                HardenedData::SingleData<real_T>     Product3_l;                    /* '<S115>/Product3' */
                                HardenedData::SingleData<real_T>     Sum_o;                         /* '<S115>/Sum' */
                                HardenedData::SingleData<real_T>     Product3_ph;                   /* '<S123>/Product3' */
                                HardenedData::SingleData<real_T>     Sum_g;                         /* '<S123>/Sum' */
                                HardenedData::SingleData<real_T>     sqrt_g;                        /* '<S122>/sqrt' */
                                HardenedData::SingleData<real_T>     Product_h;                     /* '<S117>/Product' */
                                HardenedData::SingleData<real_T>     Product1_j;                    /* '<S117>/Product1' */
                                HardenedData::SingleData<real_T>     Product2_dj;                   /* '<S117>/Product2' */
                                HardenedData::SingleData<real_T>     Product3_e;                    /* '<S117>/Product3' */
                                HardenedData::SingleData<real_T>     fcn1;                          /* '<S11>/fcn1' */
                                HardenedData::SingleData<real_T>     fcn2;                          /* '<S11>/fcn2' */
                                HardenedData::SingleData<real_T>     fcn3;                          /* '<S11>/fcn3' */
                                HardenedData::SingleData<real_T>     Merge_il;                      /* '<S118>/Merge' */
                                HardenedData::SingleData<real_T>     fcn4;                          /* '<S11>/fcn4' */
                                HardenedData::SingleData<real_T>     fcn5;                          /* '<S11>/fcn5' */
                                HardenedData::SingleData<real_T>     VectorConcatenate[3];          /* '<S116>/Vector Concatenate'
        */
                                HardenedData::SingleData<real_T>     DerivativeGain[3];             /* '<S85>/Derivative Gain' */
                                HardenedData::SingleData<real_T>     Tsamp[3];                      /* '<S88>/Tsamp' */
                                HardenedData::SingleData<real_T>     UD[3];                         /* '<S86>/UD' */
                                HardenedData::SingleData<real_T>     Diff[3];                       /* '<S86>/Diff' */
                                HardenedData::SingleData<real_T>     IntegralGain[3];               /* '<S90>/Integral Gain' */
                                HardenedData::SingleData<real_T>     Integrator[3];                 /* '<S93>/Integrator' */
                                HardenedData::SingleData<real_T>     ProportionalGain[3];           /* '<S98>/Proportional Gain'
       */
                                HardenedData::SingleData<real_T>     Product_a;                     /* '<S58>/Product' */
                                HardenedData::SingleData<real_T>     Product1_k;                    /* '<S58>/Product1' */
                                HardenedData::SingleData<real_T>     Product2_e;                    /* '<S58>/Product2' */
                                HardenedData::SingleData<real_T>     Product3_g;                    /* '<S58>/Product3' */
                                HardenedData::SingleData<real_T>     Product4;                      /* '<S58>/Product4' */
```

```
            HardenedData::SingleData<real_T>    Product5;                    /* '<S58>/Product5' */
            HardenedData::SingleData<real_T>    detMatrix;                   /* '<S58>/Sum' */
            HardenedData::SingleData<real_T>    Bias;                        /* '<S52>/Bias' */
            HardenedData::SingleData<real_T>    Abs1;                        /* '<S52>/Abs1' */
            HardenedData::SingleData<real_T>    MathFunction[9];             /* '<S51>/Math Function' */
            HardenedData::SingleData<real_T>    Product_c[9];                /* '<S51>/Product' */
            HardenedData::SingleData<real_T>    Bias1[9];                    /* '<S51>/Bias1' */
            HardenedData::SingleData<real_T>    Abs2[9];                     /* '<S51>/Abs2' */
            HardenedData::SingleData<real_T>    Add_m;                       /* '<S34>/Add' */
            HardenedData::SingleData<real_T>    sqrt_m;                      /* '<S26>/sqrt' */
            HardenedData::SingleData<real_T>    Switch[2];                   /* '<S33>/Switch' */
            HardenedData::SingleData<real_T>    Product_k;                   /* '<S33>/Product' */
            HardenedData::SingleData<real_T>    Add_o;                       /* '<S32>/Add' */
            HardenedData::SingleData<real_T>    Add_ov;                      /* '<S30>/Add' */
            HardenedData::SingleData<real_T>    Add_l;                       /* '<S31>/Add' */
            HardenedData::SingleData<real_T>    Product_cy[3];               /* '<S26>/Product' */
            HardenedData::SingleData<real_T>    Add_f;                       /* '<S44>/Add' */
            HardenedData::SingleData<real_T>    sqrt_l;                      /* '<S28>/sqrt' */
            HardenedData::SingleData<real_T>    Add_e;                       /* '<S40>/Add' */
            HardenedData::SingleData<real_T>    Add_l2;                      /* '<S41>/Add' */
            HardenedData::SingleData<real_T>    Add_ek;                      /* '<S42>/Add' */
            HardenedData::SingleData<real_T>    Switch_b[2];                 /* '<S43>/Switch' */
            HardenedData::SingleData<real_T>    Product_d;                   /* '<S43>/Product' */
            HardenedData::SingleData<real_T>    Product_i[3];                /* '<S28>/Product' */
            HardenedData::SingleData<real_T>    Add_g;                       /* '<S39>/Add' */
            HardenedData::SingleData<real_T>    sqrt_mw;                     /* '<S27>/sqrt' */
            HardenedData::SingleData<real_T>    Add_gw;                      /* '<S37>/Add' */
            HardenedData::SingleData<real_T>    Add_fc;                      /* '<S36>/Add' */
            HardenedData::SingleData<real_T>    Add_d;                       /* '<S35>/Add' */
            HardenedData::SingleData<real_T>    Switch_f[2];                 /* '<S38>/Switch' */
            HardenedData::SingleData<real_T>    Product_e;                   /* '<S38>/Product' */
            HardenedData::SingleData<real_T>    Product_o[3];                /* '<S27>/Product' */
            HardenedData::SingleData<real_T>    Sum_gh;                      /* '<S23>/Sum' */
            HardenedData::SingleData<real_T>    sqrt_a;                      /* '<S23>/sqrt' */
            HardenedData::SingleData<real_T>    Gain1;                       /* '<S23>/Gain1' */
            HardenedData::SingleData<real_T>    Add_k;                       /* '<S46>/Add' */
            HardenedData::SingleData<real_T>    Add_el;                      /* '<S45>/Add' */
            HardenedData::SingleData<real_T>    Add_a;                       /* '<S47>/Add' */
            HardenedData::SingleData<real_T>    Merge_m;                     /* '<S129>/Merge' */
            HardenedData::SingleData<real_T>    Product_n;                   /* '<S259>/Product' */
            HardenedData::SingleData<real_T>    Product1_ii;                 /* '<S259>/Product1' */
            HardenedData::SingleData<real_T>    Product2_c;                  /* '<S259>/Product2' */
            HardenedData::SingleData<real_T>    Product3_d;                  /* '<S259>/Product3' */
            HardenedData::SingleData<real_T>    Sum_j;                       /* '<S259>/Sum' */
            HardenedData::SingleData<real_T>    sqrt_n;                      /* '<S258>/sqrt' */
            HardenedData::SingleData<real_T>    Product2_oi;                 /* '<S254>/Product2' */
            HardenedData::SingleData<real_T>    Product6;                    /* '<S255>/Product6' */
            HardenedData::SingleData<real_T>    Product3_a;                  /* '<S254>/Product3' */
            HardenedData::SingleData<real_T>    Product7;                    /* '<S255>/Product7' */
            HardenedData::SingleData<real_T>    Sum3;                        /* '<S255>/Sum3' */
            HardenedData::SingleData<real_T>    Gain2;                       /* '<S255>/Gain2' */
            HardenedData::SingleData<real_T>    Product8;                    /* '<S255>/Product8' */
            HardenedData::SingleData<real_T>    Product1_fv;                 /* '<S254>/Product1' */
            HardenedData::SingleData<real_T>    Product_jb;                  /* '<S255>/Product' */
            HardenedData::SingleData<real_T>    Product_cw;                  /* '<S254>/Product' */
            HardenedData::SingleData<real_T>    Product1_b;                  /* '<S255>/Product1' */
            HardenedData::SingleData<real_T>    Sum1;                        /* '<S255>/Sum1' */
            HardenedData::SingleData<real_T>    Gain;                        /* '<S255>/Gain' */
            HardenedData::SingleData<real_T>    Product4_p;                  /* '<S255>/Product4' */
            HardenedData::SingleData<real_T>    Product2_ox;                 /* '<S255>/Product2' */
            HardenedData::SingleData<real_T>    Product3_pv;                 /* '<S255>/Product3' */
            HardenedData::SingleData<real_T>    Sum2;                        /* '<S255>/Sum2' */
            HardenedData::SingleData<real_T>    Gain1_l;                     /* '<S255>/Gain1' */
            HardenedData::SingleData<real_T>    Product5_g;                  /* '<S255>/Product5' */
            HardenedData::SingleData<real_T>    Sum_k;                       /* '<S255>/Sum' */
            HardenedData::SingleData<real_T>    Product_if;                  /* '<S256>/Product' */
            HardenedData::SingleData<real_T>    Product1_e;                  /* '<S256>/Product1' */
            HardenedData::SingleData<real_T>    Sum1_b;                      /* '<S256>/Sum1' */
            HardenedData::SingleData<real_T>    Gain_a;                      /* '<S256>/Gain' */
            HardenedData::SingleData<real_T>    Product4_k;                  /* '<S256>/Product4' */
            HardenedData::SingleData<real_T>    Product6_d;                  /* '<S256>/Product6' */
            HardenedData::SingleData<real_T>    Product7_e;                  /* '<S256>/Product7' */
            HardenedData::SingleData<real_T>    Sum3_p;                      /* '<S256>/Sum3' */
            HardenedData::SingleData<real_T>    Gain2_a;                     /* '<S256>/Gain2' */
            HardenedData::SingleData<real_T>    Product8_c;                  /* '<S256>/Product8' */
            HardenedData::SingleData<real_T>    Product2_l;                  /* '<S256>/Product2' */
            HardenedData::SingleData<real_T>    Product3_c5;                 /* '<S256>/Product3' */
            HardenedData::SingleData<real_T>    Sum2_k;                      /* '<S256>/Sum2' */
            HardenedData::SingleData<real_T>    Gain1_d;                     /* '<S256>/Gain1' */
            HardenedData::SingleData<real_T>    Product5_j;                  /* '<S256>/Product5' */
            HardenedData::SingleData<real_T>    Sum_lm;                      /* '<S256>/Sum' */
            HardenedData::SingleData<real_T>    Product_c3;                  /* '<S257>/Product' */
            HardenedData::SingleData<real_T>    Product1_c;                  /* '<S257>/Product1' */
            HardenedData::SingleData<real_T>    Sum1_bu;                     /* '<S257>/Sum1' */
            HardenedData::SingleData<real_T>    Gain_k;                      /* '<S257>/Gain' */
            HardenedData::SingleData<real_T>    Product4_m;                  /* '<S257>/Product4' */
            HardenedData::SingleData<real_T>    Product2_k;                  /* '<S257>/Product2' */
            HardenedData::SingleData<real_T>    Product3_f;                  /* '<S257>/Product3' */
            HardenedData::SingleData<real_T>    Sum2_p;                      /* '<S257>/Sum2' */
            HardenedData::SingleData<real_T>    Gain1_i;                     /* '<S257>/Gain1' */
            HardenedData::SingleData<real_T>    Product5_d;                  /* '<S257>/Product5' */
            HardenedData::SingleData<real_T>    Product6_b;                  /* '<S257>/Product6' */
            HardenedData::SingleData<real_T>    Product7_h;                  /* '<S257>/Product7' */
            HardenedData::SingleData<real_T>    Sum3_e;                      /* '<S257>/Sum3' */
            HardenedData::SingleData<real_T>    Gain2_b;                     /* '<S257>/Gain2' */
            HardenedData::SingleData<real_T>    Product8_g;                  /* '<S257>/Product8' */
            HardenedData::SingleData<real_T>    Sum_n;                       /* '<S257>/Sum' */
            HardenedData::SingleData<real_T>    TmpSignalConversionAtDotProduct[3];
            HardenedData::SingleData<real_T>    Product_jz;                  /* '<S265>/Product' */
            HardenedData::SingleData<real_T>    Product1_bp;                 /* '<S265>/Product1' */
            HardenedData::SingleData<real_T>    Product2_b;                  /* '<S265>/Product2' */
            HardenedData::SingleData<real_T>    Product3_dh;                 /* '<S265>/Product3' */
            HardenedData::SingleData<real_T>    Sum_d5;                      /* '<S265>/Sum' */
            HardenedData::SingleData<real_T>    sqrt_k;                      /* '<S264>/sqrt' */
```

```
HardenedData::SingleData<real_T>    Product2_lq;                   /* '<S260>/Product2' */
HardenedData::SingleData<real_T>    Product6_l;                    /* '<S261>/Product6' */
HardenedData::SingleData<real_T>    Product3_h;                    /* '<S260>/Product3' */
HardenedData::SingleData<real_T>    Product7_d;                    /* '<S261>/Product7' */
HardenedData::SingleData<real_T>    Sum3_j;                        /* '<S261>/Sum3' */
HardenedData::SingleData<real_T>    Gain2_f;                       /* '<S261>/Gain2' */
HardenedData::SingleData<real_T>    Product8_m;                    /* '<S261>/Product8' */
HardenedData::SingleData<real_T>    Product1_b1;                   /* '<S260>/Product1' */
HardenedData::SingleData<real_T>    Product_h1;                    /* '<S261>/Product' */
HardenedData::SingleData<real_T>    Product_f;                     /* '<S260>/Product' */
HardenedData::SingleData<real_T>    Product1_l;                    /* '<S261>/Product1' */
HardenedData::SingleData<real_T>    Sum1_d;                        /* '<S261>/Sum1' */
HardenedData::SingleData<real_T>    Gain_j;                        /* '<S261>/Gain' */
HardenedData::SingleData<real_T>    Product4_o;                    /* '<S261>/Product4' */
HardenedData::SingleData<real_T>    Product2_p;                    /* '<S261>/Product2' */
HardenedData::SingleData<real_T>    Product3_lx;                   /* '<S261>/Product3' */
HardenedData::SingleData<real_T>    Sum2_p3;                       /* '<S261>/Sum2' */
HardenedData::SingleData<real_T>    Gain1_o;                       /* '<S261>/Gain1' */
HardenedData::SingleData<real_T>    Product5_i;                    /* '<S261>/Product5' */
HardenedData::SingleData<real_T>    Sum_kk;                        /* '<S261>/Sum' */
HardenedData::SingleData<real_T>    Product_bg;                    /* '<S262>/Product' */
HardenedData::SingleData<real_T>    Product1_l3;                   /* '<S262>/Product1' */
HardenedData::SingleData<real_T>    Sum1_l;                        /* '<S262>/Sum1' */
HardenedData::SingleData<real_T>    Gain_i;                        /* '<S262>/Gain' */
HardenedData::SingleData<real_T>    Product4_pt;                   /* '<S262>/Product4' */
HardenedData::SingleData<real_T>    Product6_p;                    /* '<S262>/Product6' */
HardenedData::SingleData<real_T>    Product7_dj;                   /* '<S262>/Product7' */
HardenedData::SingleData<real_T>    Sum3_o;                        /* '<S262>/Sum3' */
HardenedData::SingleData<real_T>    Gain2_j;                       /* '<S262>/Gain2' */
HardenedData::SingleData<real_T>    Product8_e;                    /* '<S262>/Product8' */
HardenedData::SingleData<real_T>    Product2_mf;                   /* '<S262>/Product2' */
HardenedData::SingleData<real_T>    Product3_k;                    /* '<S262>/Product3' */
HardenedData::SingleData<real_T>    Sum2_a;                        /* '<S262>/Sum2' */
HardenedData::SingleData<real_T>    Gain1_oi;                      /* '<S262>/Gain1' */
HardenedData::SingleData<real_T>    Product5_b;                    /* '<S262>/Product5' */
HardenedData::SingleData<real_T>    Sum_ln;                        /* '<S262>/Sum' */
HardenedData::SingleData<real_T>    Product_l;                     /* '<S263>/Product' */
HardenedData::SingleData<real_T>    Product1_id;                   /* '<S263>/Product1' */
HardenedData::SingleData<real_T>    Sum1_d1;                       /* '<S263>/Sum1' */
HardenedData::SingleData<real_T>    Gain_o;                        /* '<S263>/Gain' */
HardenedData::SingleData<real_T>    Product4_d;                    /* '<S263>/Product4' */
HardenedData::SingleData<real_T>    Product2_f;                    /* '<S263>/Product2' */
HardenedData::SingleData<real_T>    Product3_gx;                   /* '<S263>/Product3' */
HardenedData::SingleData<real_T>    Sum2_l;                        /* '<S263>/Sum2' */
HardenedData::SingleData<real_T>    Gain1_g;                       /* '<S263>/Gain1' */
HardenedData::SingleData<real_T>    Product5_n;                    /* '<S263>/Product5' */
HardenedData::SingleData<real_T>    Product6_j;                    /* '<S263>/Product6' */
HardenedData::SingleData<real_T>    Product7_g;                    /* '<S263>/Product7' */
HardenedData::SingleData<real_T>    Sum3_b;                        /* '<S263>/Sum3' */
HardenedData::SingleData<real_T>    Gain2_o;                       /* '<S263>/Gain2' */
HardenedData::SingleData<real_T>    Product8_a;                    /* '<S263>/Product8' */
HardenedData::SingleData<real_T>    Sum_kb;                        /* '<S263>/Sum' */
HardenedData::SingleData<real_T>    TmpSignalConversionAtDotProdu_d[3];
HardenedData::SingleData<real_T>    DotProduct3;                   /* '<S130>/Dot Product3' */
HardenedData::SingleData<real_T>    DotProduct1;                   /* '<S130>/Dot Product1' */
HardenedData::SingleData<real_T>    DotProduct2;                   /* '<S130>/Dot Product2' */
HardenedData::SingleData<real_T>    Divide1_k;                     /* '<S130>/Divide1' */
HardenedData::SingleData<real_T>    Sqrt3;                         /* '<S130>/Sqrt3' */
HardenedData::SingleData<real_T>    Divide_o;                      /* '<S130>/Divide' */
HardenedData::SingleData<real_T>    Bias_k;                        /* '<S130>/Bias' */
HardenedData::SingleData<real_T>    Abs;                           /* '<S130>/Abs' */
HardenedData::SingleData<real_T>    Bias1_j;                       /* '<S130>/Bias1' */
HardenedData::SingleData<real_T>    Abs1_d;                        /* '<S130>/Abs1' */
HardenedData::SingleData<real_T>    Merge_ip;                      /* '<S130>/Merge' */
HardenedData::SingleData<real_T>    DotProduct3_o;                 /* '<S136>/Dot Product3' */
HardenedData::SingleData<real_T>    DotProduct2_c;                 /* '<S136>/Dot Product2' */
HardenedData::SingleData<real_T>    Divide1_d;                     /* '<S136>/Divide1' */
HardenedData::SingleData<real_T>    Sqrt3_j;                       /* '<S136>/Sqrt3' */
HardenedData::SingleData<real_T>    Add2;                          /* '<S136>/Add2' */
HardenedData::SingleData<real_T>    Product_i2;                    /* '<S176>/Product' */
HardenedData::SingleData<real_T>    DotProduct3_p;                 /* '<S163>/Dot Product3' */
HardenedData::SingleData<real_T>    DotProduct2_f;                 /* '<S163>/Dot Product2' */
HardenedData::SingleData<real_T>    Divide1_b;                     /* '<S163>/Divide1' */
HardenedData::SingleData<real_T>    Sqrt3_l;                       /* '<S163>/Sqrt3' */
HardenedData::SingleData<real_T>    Divide_k;                      /* '<S163>/Divide' */
HardenedData::SingleData<real_T>    Bias_b;                        /* '<S163>/Bias' */
HardenedData::SingleData<real_T>    Abs_o;                         /* '<S163>/Abs' */
HardenedData::SingleData<real_T>    Bias1_m;                       /* '<S163>/Bias1' */
HardenedData::SingleData<real_T>    Abs1_l;                        /* '<S163>/Abs1' */
HardenedData::SingleData<real_T>    Merge_d;                       /* '<S163>/Merge' */
HardenedData::SingleData<real_T>    is180degRot[3];                /* '<S136>/is 180deg Rot' */
HardenedData::SingleData<real_T>    Product1_g;                    /* '<S176>/Product1' */
HardenedData::SingleData<real_T>    Product2_g;                    /* '<S176>/Product2' */
HardenedData::SingleData<real_T>    Product3_aj;                   /* '<S176>/Product3' */
HardenedData::SingleData<real_T>    Sum_c;                         /* '<S176>/Sum' */
HardenedData::SingleData<real_T>    sqrt_h;                        /* '<S175>/sqrt' */
HardenedData::SingleData<real_T>    Product_fv;                    /* '<S166>/Product' */
HardenedData::SingleData<real_T>    Product2_a;                    /* '<S166>/Product2' */
HardenedData::SingleData<real_T>    Product_l5;                    /* '<S155>/Product' */
HardenedData::SingleData<real_T>    Product1_n;                    /* '<S166>/Product1' */
HardenedData::SingleData<real_T>    Product1_lg;                   /* '<S155>/Product1' */
HardenedData::SingleData<real_T>    Product2_fh;                   /* '<S155>/Product2' */
HardenedData::SingleData<real_T>    Product3_pe;                   /* '<S166>/Product3' */
HardenedData::SingleData<real_T>    Product3_l0;                   /* '<S155>/Product3' */
HardenedData::SingleData<real_T>    Sum_n3;                        /* '<S155>/Sum' */
HardenedData::SingleData<real_T>    sqrt_mv;                       /* '<S154>/sqrt' */
HardenedData::SingleData<real_T>    Product2_ke;                   /* '<S150>/Product2' */
HardenedData::SingleData<real_T>    Product6_e;                    /* '<S151>/Product6' */
HardenedData::SingleData<real_T>    Product3_cc;                   /* '<S150>/Product3' */
HardenedData::SingleData<real_T>    Product7_a;                    /* '<S151>/Product7' */
HardenedData::SingleData<real_T>    Sum3_b5;                       /* '<S151>/Sum3' */
HardenedData::SingleData<real_T>    Gain2_p;                       /* '<S151>/Gain2' */
HardenedData::SingleData<real_T>    Product8_k;                    /* '<S151>/Product8' */
HardenedData::SingleData<real_T>    Product1_l0;                   /* '<S150>/Product1' */
```

```
                    HardenedData::SingleData<real_T>    Product_da;                     /* '<S151>/Product' */
                    HardenedData::SingleData<real_T>    Product_m;                      /* '<S150>/Product' */
                    HardenedData::SingleData<real_T>    Product1_dj;                    /* '<S151>/Product1' */
                    HardenedData::SingleData<real_T>    Sum1_n;                         /* '<S151>/Sum1' */
                    HardenedData::SingleData<real_T>    Gain_l;                         /* '<S151>/Gain' */
                    HardenedData::SingleData<real_T>    Product4_c;                     /* '<S151>/Product4' */
                    HardenedData::SingleData<real_T>    Product2_gk;                    /* '<S151>/Product2' */
                    HardenedData::SingleData<real_T>    Product3_i;                     /* '<S151>/Product3' */
                    HardenedData::SingleData<real_T>    Sum2_g;                         /* '<S151>/Sum2' */
                    HardenedData::SingleData<real_T>    Gain1_e;                        /* '<S151>/Gain1' */
                    HardenedData::SingleData<real_T>    Product5_o;                     /* '<S151>/Product5' */
                    HardenedData::SingleData<real_T>    Sum_oz;                         /* '<S151>/Sum' */
                    HardenedData::SingleData<real_T>    Product_g;                      /* '<S152>/Product' */
                    HardenedData::SingleData<real_T>    Product1_ew;                    /* '<S152>/Product1' */
                    HardenedData::SingleData<real_T>    Sum1_m;                         /* '<S152>/Sum1' */
                    HardenedData::SingleData<real_T>    Gain_b;                         /* '<S152>/Gain' */
                    HardenedData::SingleData<real_T>    Product4_a;                     /* '<S152>/Product4' */
                    HardenedData::SingleData<real_T>    Product6_o;                     /* '<S152>/Product6' */
                    HardenedData::SingleData<real_T>    Product7_k;                     /* '<S152>/Product7' */
                    HardenedData::SingleData<real_T>    Sum3_bi;                        /* '<S152>/Sum3' */
                    HardenedData::SingleData<real_T>    Gain2_l;                        /* '<S152>/Gain2' */
                    HardenedData::SingleData<real_T>    Product8_me;                    /* '<S152>/Product8' */
                    HardenedData::SingleData<real_T>    Product2_fhq;                   /* '<S152>/Product2' */
                    HardenedData::SingleData<real_T>    Product3_n;                     /* '<S152>/Product3' */
                    HardenedData::SingleData<real_T>    Sum2_la;                        /* '<S152>/Sum2' */
                    HardenedData::SingleData<real_T>    Gain1_ox;                       /* '<S152>/Gain1' */
                    HardenedData::SingleData<real_T>    Product5_l;                     /* '<S152>/Product5' */
                    HardenedData::SingleData<real_T>    Sum_oa;                         /* '<S152>/Sum' */
                    HardenedData::SingleData<real_T>    Product_ew;                     /* '<S153>/Product' */
                    HardenedData::SingleData<real_T>    Product1_m;                     /* '<S153>/Product1' */
                    HardenedData::SingleData<real_T>    Sum1_h;                         /* '<S153>/Sum1' */
                    HardenedData::SingleData<real_T>    Gain_kp;                        /* '<S153>/Gain' */
                    HardenedData::SingleData<real_T>    Product4_g;                     /* '<S153>/Product4' */
                    HardenedData::SingleData<real_T>    Product2_gh;                    /* '<S153>/Product2' */
                    HardenedData::SingleData<real_T>    Product3_k4;                    /* '<S153>/Product3' */
                    HardenedData::SingleData<real_T>    Sum2_i;                         /* '<S153>/Sum2' */
                    HardenedData::SingleData<real_T>    Gain1_f;                        /* '<S153>/Gain1' */
                    HardenedData::SingleData<real_T>    Product5_ij;                    /* '<S153>/Product5' */
                    HardenedData::SingleData<real_T>    Product6_c;                     /* '<S153>/Product6' */
                    HardenedData::SingleData<real_T>    Product7_a4;                    /* '<S153>/Product7' */
                    HardenedData::SingleData<real_T>    Sum3_d;                         /* '<S153>/Sum3' */
                    HardenedData::SingleData<real_T>    Gain2_n;                        /* '<S153>/Gain2' */
                    HardenedData::SingleData<real_T>    Product8_f;                     /* '<S153>/Product8' */
                    HardenedData::SingleData<real_T>    Sum_nx;                         /* '<S153>/Sum' */
                    HardenedData::SingleData<real_T>    TmpSignalConversionAtDotProdu_k[3];
                    HardenedData::SingleData<real_T>    jxk_b;                          /* '<S200>/j x k' */
                    HardenedData::SingleData<real_T>    kxi_b;                          /* '<S200>/k x i' */
                    HardenedData::SingleData<real_T>    ixj_h;                          /* '<S200>/i x j' */
                    HardenedData::SingleData<real_T>    kxj_l;                          /* '<S201>/k x j' */
                    HardenedData::SingleData<real_T>    ixk_e;                          /* '<S201>/i x k' */
                    HardenedData::SingleData<real_T>    jxi_n;                          /* '<S201>/j x i' */
                    HardenedData::SingleData<real_T>    Sum_m[3];                       /* '<S198>/Sum' */
                    HardenedData::SingleData<real_T>    jxk_e;                          /* '<S202>/j x k' */
                    HardenedData::SingleData<real_T>    kxi_es;                         /* '<S202>/k x i' */
                    HardenedData::SingleData<real_T>    ixj_n;                          /* '<S202>/i x j' */
                    HardenedData::SingleData<real_T>    kxj_j;                          /* '<S203>/k x j' */
                    HardenedData::SingleData<real_T>    ixk_j;                          /* '<S203>/i x k' */
                    HardenedData::SingleData<real_T>    jxi_f;                          /* '<S203>/j x i' */
                    HardenedData::SingleData<real_T>    Sum_pb[3];                      /* '<S199>/Sum' */
                    HardenedData::SingleData<real_T>    Product_dj;                     /* '<S161>/Product' */
                    HardenedData::SingleData<real_T>    Product1_ig;                    /* '<S161>/Product1' */
                    HardenedData::SingleData<real_T>    Product2_ad;                    /* '<S161>/Product2' */
                    HardenedData::SingleData<real_T>    Product3_hf;                    /* '<S161>/Product3' */
                    HardenedData::SingleData<real_T>    Sum_gt;                         /* '<S161>/Sum' */
                    HardenedData::SingleData<real_T>    sqrt_i;                         /* '<S160>/sqrt' */
                    HardenedData::SingleData<real_T>    Product2_nx;                    /* '<S156>/Product2' */
                    HardenedData::SingleData<real_T>    Product6_a;                     /* '<S157>/Product6' */
                    HardenedData::SingleData<real_T>    Product3_ce;                    /* '<S156>/Product3' */
                    HardenedData::SingleData<real_T>    Product7_ha;                    /* '<S157>/Product7' */
                    HardenedData::SingleData<real_T>    Sum3_l;                         /* '<S157>/Sum3' */
                    HardenedData::SingleData<real_T>    Gain2_om;                       /* '<S157>/Gain2' */
                    HardenedData::SingleData<real_T>    Product8_p;                     /* '<S157>/Product8' */
                    HardenedData::SingleData<real_T>    Product1_a;                     /* '<S156>/Product1' */
                    HardenedData::SingleData<real_T>    Product_mi;                     /* '<S157>/Product' */
                    HardenedData::SingleData<real_T>    Product_pb;                     /* '<S156>/Product' */
                    HardenedData::SingleData<real_T>    Product1_lr;                    /* '<S157>/Product1' */
                    HardenedData::SingleData<real_T>    Sum1_a;                         /* '<S157>/Sum1' */
                    HardenedData::SingleData<real_T>    Gain_p;                         /* '<S157>/Gain' */
                    HardenedData::SingleData<real_T>    Product4_i;                     /* '<S157>/Product4' */
                    HardenedData::SingleData<real_T>    Product2_nz;                    /* '<S157>/Product2' */
                    HardenedData::SingleData<real_T>    Product3_lu;                    /* '<S157>/Product3' */
                    HardenedData::SingleData<real_T>    Sum2_m;                         /* '<S157>/Sum2' */
                    HardenedData::SingleData<real_T>    Gain1_c;                        /* '<S157>/Gain1' */
                    HardenedData::SingleData<real_T>    Product5_jx;                    /* '<S157>/Product5' */
                    HardenedData::SingleData<real_T>    Sum_g5;                         /* '<S157>/Sum' */
                    HardenedData::SingleData<real_T>    Product_pe;                     /* '<S158>/Product' */
                    HardenedData::SingleData<real_T>    Product1_mu;                    /* '<S158>/Product1' */
                    HardenedData::SingleData<real_T>    Sum1_k;                         /* '<S158>/Sum1' */
                    HardenedData::SingleData<real_T>    Gain_ay;                        /* '<S158>/Gain' */
                    HardenedData::SingleData<real_T>    Product4_ok;                    /* '<S158>/Product4' */
                    HardenedData::SingleData<real_T>    Product6_ld;                    /* '<S158>/Product6' */
                    HardenedData::SingleData<real_T>    Product7_ei;                    /* '<S158>/Product7' */
                    HardenedData::SingleData<real_T>    Sum3_h;                         /* '<S158>/Sum3' */
                    HardenedData::SingleData<real_T>    Gain2_l2;                       /* '<S158>/Gain2' */
                    HardenedData::SingleData<real_T>    Product8_d;                     /* '<S158>/Product8' */
                    HardenedData::SingleData<real_T>    Product2_ea;                    /* '<S158>/Product2' */
                    HardenedData::SingleData<real_T>    Product3_ll;                    /* '<S158>/Product3' */
                    HardenedData::SingleData<real_T>    Sum2_mr;                        /* '<S158>/Sum2' */
                    HardenedData::SingleData<real_T>    Gain1_n;                        /* '<S158>/Gain1' */
                    HardenedData::SingleData<real_T>    Product5_a;                     /* '<S158>/Product5' */
                    HardenedData::SingleData<real_T>    Sum_gz;                         /* '<S158>/Sum' */
                    HardenedData::SingleData<real_T>    Product_am;                     /* '<S159>/Product' */
                    HardenedData::SingleData<real_T>    Product1_bt;                    /* '<S159>/Product1' */
                    HardenedData::SingleData<real_T>    Sum1_hk;                        /* '<S159>/Sum1' */
```

```
    HardenedData::SingleData<real_T>    Gain_g;                          /* '<S159>/Gain' */
    HardenedData::SingleData<real_T>    Product4_o5;                     /* '<S159>/Product4' */
    HardenedData::SingleData<real_T>    Product2_cc;                     /* '<S159>/Product2' */
    HardenedData::SingleData<real_T>    Product3_cd;                     /* '<S159>/Product3' */
    HardenedData::SingleData<real_T>    Sum2_o;                          /* '<S159>/Sum2' */
    HardenedData::SingleData<real_T>    Gain1_ni;                        /* '<S159>/Gain1' */
    HardenedData::SingleData<real_T>    Product5_n2;                     /* '<S159>/Product5' */
    HardenedData::SingleData<real_T>    Product6_d5;                     /* '<S159>/Product6' */
    HardenedData::SingleData<real_T>    Product7_em;                     /* '<S159>/Product7' */
    HardenedData::SingleData<real_T>    Sum3_jh;                         /* '<S159>/Sum3' */
    HardenedData::SingleData<real_T>    Gain2_ja;                        /* '<S159>/Gain2' */
    HardenedData::SingleData<real_T>    Product8_j;                      /* '<S159>/Product8' */
    HardenedData::SingleData<real_T>    Sum_pw;                          /* '<S159>/Sum' */
    HardenedData::SingleData<real_T>    TmpSignalConversionAtDotProdu_j[3];
    HardenedData::SingleData<real_T>    DotProduct3_ph;                  /* '<S178>/Dot Product3' */
    HardenedData::SingleData<real_T>    DotProduct1_o;                   /* '<S178>/Dot Product1' */
    HardenedData::SingleData<real_T>    DotProduct2_g;                   /* '<S178>/Dot Product2' */
    HardenedData::SingleData<real_T>    Divide1_p;                       /* '<S178>/Divide1' */
    HardenedData::SingleData<real_T>    Sqrt3_m;                         /* '<S178>/Sqrt3' */
    HardenedData::SingleData<real_T>    Divide_c;                        /* '<S178>/Divide' */
    HardenedData::SingleData<real_T>    Bias_h;                          /* '<S178>/Bias' */
    HardenedData::SingleData<real_T>    Abs_m;                           /* '<S178>/Abs' */
    HardenedData::SingleData<real_T>    Bias1_e;                         /* '<S178>/Bias1' */
    HardenedData::SingleData<real_T>    Abs1_i;                          /* '<S178>/Abs1' */
    HardenedData::SingleData<real_T>    Merge_l;                         /* '<S178>/Merge' */
    HardenedData::SingleData<real_T>    Switch_c[4];                     /* '<S124>/Switch' */
    HardenedData::SingleData<real_T>    Product_h0;                      /* '<S273>/Product' */
    HardenedData::SingleData<real_T>    Product1_fm;                     /* '<S273>/Product1' */
    HardenedData::SingleData<real_T>    Product2_c2;                     /* '<S273>/Product2' */
    HardenedData::SingleData<real_T>    Product3_pq;                     /* '<S273>/Product3' */
    HardenedData::SingleData<real_T>    Sum_f;                           /* '<S273>/Sum' */
    HardenedData::SingleData<real_T>    sqrt_mo;                         /* '<S272>/sqrt' */
    HardenedData::SingleData<real_T>    Product_jc;                      /* '<S267>/Product' */
    HardenedData::SingleData<real_T>    Product1_o;                      /* '<S267>/Product1' */
    HardenedData::SingleData<real_T>    Product2_j;                      /* '<S267>/Product2' */
    HardenedData::SingleData<real_T>    Product3_lz;                     /* '<S267>/Product3' */
    HardenedData::SingleData<real_T>    fcn1_i;                          /* '<S128>/fcn1' */
    HardenedData::SingleData<real_T>    fcn2_f;                          /* '<S128>/fcn2' */
    HardenedData::SingleData<real_T>    fcn3_o;                          /* '<S128>/fcn3' */
    HardenedData::SingleData<real_T>    Merge_k;                         /* '<S268>/Merge' */
    HardenedData::SingleData<real_T>    fcn4_n;                          /* '<S128>/fcn4' */
    HardenedData::SingleData<real_T>    fcn5_g;                          /* '<S128>/fcn5' */
    HardenedData::SingleData<real_T>    VectorConcatenate_a[3];          /* '<S266>/Vector Concatenate'
       */
    HardenedData::SingleData<real_T>    DerivativeGain_k[3];             /* '<S229>/Derivative Gain' */
    HardenedData::SingleData<real_T>    Tsamp_f[3];                      /* '<S232>/Tsamp' */
    HardenedData::SingleData<real_T>    UD_l[3];                         /* '<S230>/UD' */
    HardenedData::SingleData<real_T>    Diff_e[3];                       /* '<S230>/Diff' */
    HardenedData::SingleData<real_T>    IntegralGain_p[3];               /* '<S234>/Integral Gain' */
    HardenedData::SingleData<real_T>    Integrator_k[3];                 /* '<S237>/Integrator' */
    HardenedData::SingleData<real_T>    ProportionalGain_h[3];           /* '<S242>/Proportional Gain'
       */
    HardenedData::SingleData<real_T>    is180degRot_l[3];                /* '<S137>/is 180deg Rot' */
    HardenedData::SingleData<real_T>    Product3_dz;                     /* '<S191>/Product3' */
    HardenedData::SingleData<real_T>    Product2_eq;                     /* '<S191>/Product2' */
    HardenedData::SingleData<real_T>    Product1_aa;                     /* '<S191>/Product1' */
    HardenedData::SingleData<real_T>    DotProduct2_n;                   /* '<S137>/Dot Product2' */
    HardenedData::SingleData<real_T>    DotProduct1_h;                   /* '<S137>/Dot Product1' */
    HardenedData::SingleData<real_T>    Divide1_h;                       /* '<S137>/Divide1' */
    HardenedData::SingleData<real_T>    Sqrt3_mq;                        /* '<S137>/Sqrt3' */
    HardenedData::SingleData<real_T>    DotProduct3_e;                   /* '<S137>/Dot Product3' */
    HardenedData::SingleData<real_T>    Add2_k;                          /* '<S137>/Add2' */
    HardenedData::SingleData<real_T>    Product_er;                      /* '<S191>/Product' */
    HardenedData::SingleData<real_T>    Sum_e;                           /* '<S191>/Sum' */
    HardenedData::SingleData<real_T>    sqrt_j;                          /* '<S190>/sqrt' */
    HardenedData::SingleData<real_T>    Product3_lzz;                    /* '<S181>/Product3' */
    HardenedData::SingleData<real_T>    Product3_ay;                     /* '<S146>/Product3' */
    HardenedData::SingleData<real_T>    Product2_ep;                     /* '<S181>/Product2' */
    HardenedData::SingleData<real_T>    Product2_bc;                     /* '<S146>/Product2' */
    HardenedData::SingleData<real_T>    Product1_ge;                     /* '<S181>/Product1' */
    HardenedData::SingleData<real_T>    Product1_fh;                     /* '<S146>/Product1' */
    HardenedData::SingleData<real_T>    Product_ey;                      /* '<S181>/Product' */
    HardenedData::SingleData<real_T>    Product_d0;                      /* '<S146>/Product' */
    HardenedData::SingleData<real_T>    Sum_ki;                          /* '<S146>/Sum' */
    HardenedData::SingleData<real_T>    Product3_aw;                     /* '<S147>/Product3' */
    HardenedData::SingleData<real_T>    Product2_pu;                     /* '<S147>/Product2' */
    HardenedData::SingleData<real_T>    Product1_b0;                     /* '<S147>/Product1' */
    HardenedData::SingleData<real_T>    Product_cq;                      /* '<S147>/Product' */
    HardenedData::SingleData<real_T>    Sum_oq;                          /* '<S147>/Sum' */
    HardenedData::SingleData<real_T>    Product3_h5;                     /* '<S148>/Product3' */
    HardenedData::SingleData<real_T>    Product2_i;                      /* '<S148>/Product2' */
    HardenedData::SingleData<real_T>    Product1_ai;                     /* '<S148>/Product1' */
    HardenedData::SingleData<real_T>    Product_cf;                      /* '<S148>/Product' */
    HardenedData::SingleData<real_T>    Sum_kc;                          /* '<S148>/Sum' */
    HardenedData::SingleData<real_T>    Product3_c1;                     /* '<S149>/Product3' */
    HardenedData::SingleData<real_T>    Product2_no;                     /* '<S149>/Product2' */
    HardenedData::SingleData<real_T>    Product1_d1;                     /* '<S149>/Product1' */
    HardenedData::SingleData<real_T>    Product_gr;                      /* '<S149>/Product' */
    HardenedData::SingleData<real_T>    Sum_n1;                          /* '<S149>/Sum' */
    HardenedData::SingleData<real_T>    Switch3[3];                      /* '<S180>/Switch3' */
    HardenedData::SingleData<real_T>    jxi_m;                           /* '<S189>/j x i' */
    HardenedData::SingleData<real_T>    ixk_k;                           /* '<S189>/i x k' */
    HardenedData::SingleData<real_T>    kxj_f;                           /* '<S189>/k x j' */
    HardenedData::SingleData<real_T>    ixj_k;                           /* '<S188>/i x j' */
    HardenedData::SingleData<real_T>    kxi_bj;                          /* '<S188>/k x i' */
    HardenedData::SingleData<real_T>    jxk_l;                           /* '<S188>/j x k' */
    HardenedData::SingleData<real_T>    Sum_pd[3];                       /* '<S187>/Sum' */
    HardenedData::SingleData<real_T>    Gain2_c;                         /* '<S180>/Gain2' */
    HardenedData::SingleData<real_T>    Gain_o2;                         /* '<S180>/Gain' */
    HardenedData::SingleData<real_T>    jxi_no;                          /* '<S183>/j x i' */
    HardenedData::SingleData<real_T>    ixk_b;                           /* '<S183>/i x k' */
    HardenedData::SingleData<real_T>    kxj_n;                           /* '<S183>/k x j' */
    HardenedData::SingleData<real_T>    ixj_l;                           /* '<S182>/i x j' */
    HardenedData::SingleData<real_T>    kxi_o;                           /* '<S182>/k x i' */
```

```
                                HardenedData::SingleData<real_T>    jxk_d;                          /* '<S182>/j x k' */
                                HardenedData::SingleData<real_T>    Sum_b[3];                       /* '<S177>/Sum' */
                                HardenedData::SingleData<real_T>    jxi_k;                          /* '<S168>/j x i' */
                                HardenedData::SingleData<real_T>    ixk_a;                          /* '<S168>/i x k' */
                                HardenedData::SingleData<real_T>    kxj_c;                          /* '<S168>/k x j' */
                                HardenedData::SingleData<real_T>    ixj_o;                          /* '<S167>/i x j' */
                                HardenedData::SingleData<real_T>    kxi_p;                          /* '<S167>/k x i' */
                                HardenedData::SingleData<real_T>    jxk_er;                         /* '<S167>/j x k' */
                                HardenedData::SingleData<real_T>    Sum_ff[3];                      /* '<S162>/Sum' */
                            HardenedData::SingleData<real_T>    Dipole_x[3];                    /* '<S2>/Magnetroquer control' */
                            HardenedData::SingleData<real_T>    Dipole_y[3];                    /* '<S2>/Magnetroquer control' */
                            HardenedData::SingleData<real_T>    Dipole_z[3];                    /* '<S2>/Magnetroquer control' */
                            HardenedData::SingleData<real_T>    bdot[3];                        /* '<S2>/Derivative Function' */
#endif
  boolean_T Compare;                   /* '<S59>/Compare' */
  boolean_T Compare_h[9];              /* '<S57>/Compare' */
  boolean_T LogicalOperator1;          /* '<S51>/Logical Operator1' */
  boolean_T Compare_k;                 /* '<S131>/Compare' */
  boolean_T Compare_a;                 /* '<S132>/Compare' */
  boolean_T OR;                        /* '<S124>/OR' */
  boolean_T Compare_m;                 /* '<S164>/Compare' */
  boolean_T Compare_g;                 /* '<S179>/Compare' */
  boolean_T xz;                        /* '<S180>/x>z' */
} B_Vehicle_T;

/* Block states (default storage) for system '<Root>' */
typedef struct {
#ifdef RAD_HARD
    HardenedData::TripleData<real_T>  UD_DSTATE[3];                 /* '<S86>/UD' */
    HardenedData::TripleData<real_T>  Integrator_DSTATE[3];         /* '<S93>/Integrator' */
    HardenedData::TripleData<real_T>  UD_DSTATE_d[3];               /* '<S230>/UD' */
    HardenedData::TripleData<real_T>  Integrator_DSTATE_f[3];       /* '<S237>/Integrator' */
    HardenedData::TripleData<real_T>  state[3];                     /* '<S2>/Derivative Function' */
#else
    HardenedData::SingleData<real_T>  UD_DSTATE[3];                 /* '<S86>/UD' */
    HardenedData::SingleData<real_T>  Integrator_DSTATE[3];         /* '<S93>/Integrator' */
    HardenedData::SingleData<real_T>  UD_DSTATE_d[3];               /* '<S230>/UD' */
    HardenedData::SingleData<real_T>  Integrator_DSTATE_f[3];       /* '<S237>/Integrator' */
    HardenedData::SingleData<real_T>  state[3];                     /* '<S2>/Derivative Function' */
#endif
} DW_Vehicle_T;

/* Invariant block signals (default storage) */
typedef struct {
#ifdef RAD_HARD
    HardenedData::TripleData<real_T>   DotProduct3;                 /* '<S129>/Dot Product3' */
                HardenedData::TripleData<real_T>   DotProduct1;                 /* '<S129>/Dot Product1' */
                HardenedData::TripleData<real_T>   DotProduct2;                 /* '<S129>/Dot Product2' */
                HardenedData::TripleData<real_T>   Divide1;                     /* '<S129>/Divide1' */
                HardenedData::TripleData<real_T>   Sqrt3;                       /* '<S129>/Sqrt3' */
                HardenedData::TripleData<real_T>   Divide;                      /* '<S129>/Divide' */
                HardenedData::TripleData<real_T>   Bias;                        /* '<S129>/Bias' */
                HardenedData::TripleData<real_T>   Abs;                         /* '<S129>/Abs' */
                HardenedData::TripleData<real_T>   Bias1;                       /* '<S129>/Bias1' */
                HardenedData::TripleData<real_T>   Abs1;                        /* '<S129>/Abs1' */
                HardenedData::TripleData<real_T>   DotProduct1_g;               /* '<S163>/Dot Product1' */
                HardenedData::TripleData<real_T>   DotProduct1_c;               /* '<S136>/Dot Product1' */
                HardenedData::TripleData<real_T>   Gain;                        /* '<S165>/Gain' */
                HardenedData::TripleData<real_T>   Gain2;                       /* '<S165>/Gain2' */
                HardenedData::TripleData<real_T>   Switch3[3];                  /* '<S165>/Switch3' */
                HardenedData::TripleData<real_T>   ixj;                         /* '<S173>/i x j' */
                HardenedData::TripleData<real_T>   jxk;                         /* '<S173>/j x k' */
                HardenedData::TripleData<real_T>   kxi;                         /* '<S173>/k x i' */
                HardenedData::TripleData<real_T>   ixk;                         /* '<S174>/i x k' */
                HardenedData::TripleData<real_T>   jxi;                         /* '<S174>/j x i' */
                HardenedData::TripleData<real_T>   kxj;                         /* '<S174>/k x j' */
                HardenedData::TripleData<real_T>   Sum[3];                      /* '<S172>/Sum' */
                HardenedData::TripleData<real_T>   ixj_g;                       /* '<S194>/i x j' */
                HardenedData::TripleData<real_T>   jxk_e;                       /* '<S194>/j x k' */
                HardenedData::TripleData<real_T>   kxi_c;                       /* '<S194>/k x i' */
                HardenedData::TripleData<real_T>   ixk_d;                       /* '<S195>/i x k' */
                HardenedData::TripleData<real_T>   jxi_e;                       /* '<S195>/j x i' */
                HardenedData::TripleData<real_T>   kxj_o;                       /* '<S195>/k x j' */
                HardenedData::TripleData<real_T>   Sum_d[3];                    /* '<S192>/Sum' */
                HardenedData::TripleData<real_T>   ixj_a;                       /* '<S196>/i x j' */
                HardenedData::TripleData<real_T>   jxk_n;                       /* '<S196>/j x k' */
                HardenedData::TripleData<real_T>   kxi_p;                       /* '<S196>/k x i' */
                HardenedData::TripleData<real_T>   ixk_c;                       /* '<S197>/i x k' */
                HardenedData::TripleData<real_T>   jxi_o;                       /* '<S197>/j x i' */
                HardenedData::TripleData<real_T>   kxj_n;                       /* '<S197>/k x j' */
                HardenedData::TripleData<real_T>   Sum_d0[3];                   /* '<S193>/Sum' */
#else
                HardenedData::SingleData<real_T>   DotProduct3;                 /* '<S129>/Dot Product3' */
                            HardenedData::SingleData<real_T>   DotProduct1;                 /* '<S129>/Dot Product1' */
                            HardenedData::SingleData<real_T>   DotProduct2;                 /* '<S129>/Dot Product2' */
                            HardenedData::SingleData<real_T>   Divide1;                     /* '<S129>/Divide1' */
                            HardenedData::SingleData<real_T>   Sqrt3;                       /* '<S129>/Sqrt3' */
                            HardenedData::SingleData<real_T>   Divide;                      /* '<S129>/Divide' */
                            HardenedData::SingleData<real_T>   Bias;                        /* '<S129>/Bias' */
                            HardenedData::SingleData<real_T>   Abs;                         /* '<S129>/Abs' */
                            HardenedData::SingleData<real_T>   Bias1;                       /* '<S129>/Bias1' */
                            HardenedData::SingleData<real_T>   Abs1;                        /* '<S129>/Abs1' */
                            HardenedData::SingleData<real_T>   DotProduct1_g;               /* '<S163>/Dot Product1' */
                            HardenedData::SingleData<real_T>   DotProduct1_c;               /* '<S136>/Dot Product1' */
                            HardenedData::SingleData<real_T>   Gain;                        /* '<S165>/Gain' */
                            HardenedData::SingleData<real_T>   Gain2;                       /* '<S165>/Gain2' */
                            HardenedData::SingleData<real_T>   Switch3[3];                  /* '<S165>/Switch3' */
                            HardenedData::SingleData<real_T>   ixj;                         /* '<S173>/i x j' */
                            HardenedData::SingleData<real_T>   jxk;                         /* '<S173>/j x k' */
                            HardenedData::SingleData<real_T>   kxi;                         /* '<S173>/k x i' */
                            HardenedData::SingleData<real_T>   ixk;                         /* '<S174>/i x k' */
                            HardenedData::SingleData<real_T>   jxi;                         /* '<S174>/j x i' */
                            HardenedData::SingleData<real_T>   kxj;                         /* '<S174>/k x j' */
                            HardenedData::SingleData<real_T>   Sum[3];                      /* '<S172>/Sum' */
```

```
                        HardenedData::SingleData<real_T>   ixj_g;                          /* '<S194>/i x j' */
                        HardenedData::SingleData<real_T>   jxk_e;                          /* '<S194>/j x k' */
                        HardenedData::SingleData<real_T>   kxi_c;                          /* '<S194>/k x i' */
                        HardenedData::SingleData<real_T>   ixk_d;                          /* '<S195>/i x k' */
                        HardenedData::SingleData<real_T>   jxi_e;                          /* '<S195>/j x i' */
                        HardenedData::SingleData<real_T>   kxj_o;                          /* '<S195>/k x j' */
                        HardenedData::SingleData<real_T>   Sum_d[3];                       /* '<S192>/Sum' */
                        HardenedData::SingleData<real_T>   ixj_a;                          /* '<S196>/i x j' */
                        HardenedData::SingleData<real_T>   jxk_n;                          /* '<S196>/j x k' */
                        HardenedData::SingleData<real_T>   kxi_p;                          /* '<S196>/k x i' */
                        HardenedData::SingleData<real_T>   ixk_c;                          /* '<S197>/i x k' */
                        HardenedData::SingleData<real_T>   jxi_o;                          /* '<S197>/j x i' */
                        HardenedData::SingleData<real_T>   kxj_n;                          /* '<S197>/k x j' */
                        HardenedData::SingleData<real_T>   Sum_d0[3];                      /* '<S193>/Sum' */
#endif
  boolean_T xz;                        /* '<S165>/x>z' */
} ConstB_Vehicle_T;

/* External inputs (root inport signals with default storage) */
typedef struct {
#ifdef RAD_HARD
    HardenedData::TripleData<real_T>   Bmgm[3];
    HardenedData::TripleData<real_T>   q_eci2b[4];                  /* '<Root>/q_eci2b' */
    HardenedData::TripleData<real_T>   xSunECI[3];                  /* '<Root>/xSunECI' */
    HardenedData::TripleData<real_T>   qECEF2b[4];                  /* '<Root>/qECEF2b' */
    HardenedData::TripleData<real_T>   vECEF[3];                    /* '<Root>/vECEF' */
    HardenedData::TripleData<real_T>  xECEF[3];                    /* '<Root>/xECEF' */
    HardenedData::TripleData<real_T>   Mode;                        /* '<Root>/Mode' */
#else
    HardenedData::SingleData<real_T>   Bmgm[3];
        HardenedData::SingleData<real_T>   q_eci2b[4];                  /* '<Root>/q_eci2b' */
        HardenedData::SingleData<real_T>   xSunECI[3];                  /* '<Root>/xSunECI' */
        HardenedData::SingleData<real_T>   qECEF2b[4];                  /* '<Root>/qECEF2b' */
        HardenedData::SingleData<real_T>   vECEF[3];                    /* '<Root>/vECEF' */
        HardenedData::SingleData<real_T>  xECEF[3];                    /* '<Root>/xECEF' */
        HardenedData::SingleData<real_T>   Mode;                        /* '<Root>/Mode' */
#endif
} ExtU_Vehicle_T;

/* External outputs (root outports fed by signals with default storage) */
typedef struct {
#ifdef RAD_HARD
    HardenedData::TripleData<real_T>  Commands[3];                  /* '<Root>/Commands' */
    HardenedData::TripleData<real_T>  Bbody[3];                     /* '<Root>/Bbody' */
#else
    HardenedData::SingleData<real_T>  Commands[3];                  /* '<Root>/Commands' */
    HardenedData::SingleData<real_T>  Bbody[3];                     /* '<Root>/Bbody' */
#endif
} ExtY_Vehicle_T;



/* Block signals (default storage) */
#ifdef __cplusplus

extern "C" {

#endif



#ifdef __cplusplus

}
#endif


#ifdef __cplusplus

extern "C" {

#endif



#ifdef __cplusplus

}
#endif

//extern const ConstB_Vehicle_T Vehicle_ConstB;/* constant block i/o */

#ifdef __cplusplus

extern "C" {

#endif

  /* Model entry point functions */
  extern void Vehicle_initialize();
  extern void Vehicle_step(void);
  extern void Vehicle_terminate(void);
  extern 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]);



  extern real_T rt_atan2d_snf(real_T u0, real_T u1);

  extern void Detumble();
  extern void NadirPointing();
  extern void SunPointing();
```

```
  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 inport 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;



#ifdef __cplusplus

}
#endif

/* Real-time Model object */
#ifdef __cplusplus

extern "C" {

#endif

 // extern RT_MODEL_Vehicle_T *const Vehicle_M;

#ifdef __cplusplus

}
#endif

#endif                                 /* RTW_HEADER_Vehicle_h_ */
```

RadiationEffects.cpp

```
#include "Vehicle.h"

enum {
    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*/
```

```
        case 1:         Vehicle_U.Bmgm[0].a()    = (real_T)index; break; //random number
        case 2:         Vehicle_U.Bmgm[1].a()    = (real_T)index; break;
        case 3:         Vehicle_U.Bmgm[2].a()    = (real_T)index; break;
        case 4:         Vehicle_U.Mode.a()       = (real_T) index; break;
        case 5:         Vehicle_U.qECEF2b[0].a() = (real_T) index; break;
        case 6:         Vehicle_U.qECEF2b[1].a() = (real_T) index; break;
        case 7:         Vehicle_U.qECEF2b[2].a() = (real_T) index; break;
        case 8:         Vehicle_U.qECEF2b[3].a() = (real_T) index; break;
        case 9:         Vehicle_U.q_eci2b[0].a() = (real_T) index; break;
        case 10:        Vehicle_U.q_eci2b[1].a() = (real_T) index; break;
        case 11:        Vehicle_U.q_eci2b[2].a() = (real_T) index; break;
        case 12:        Vehicle_U.q_eci2b[3].a() = (real_T) index; break;
        case 13:        Vehicle_U.vECEF[0].a()   = (real_T) index; break;
        case 14:        Vehicle_U.vECEF[1].a()   = (real_T) index; break;
        case 15:        Vehicle_U.vECEF[2].a()   = (real_T) index; break;
        case 16:        Vehicle_U.xECEF[0].a()   = (real_T) index; break;
        case 17:        Vehicle_U.xECEF[1].a()   = (real_T) index; break;
        case 18:        Vehicle_U.xECEF[2].a()   = (real_T) index; break;
        case 19:        Vehicle_U.xSunECI[0].a() = (real_T) index; break;
        case 20:        Vehicle_U.xSunECI[1].a() = (real_T) index; break;
        case 21:        Vehicle_U.xSunECI[2].a() = (real_T) index; break;
        }
}

/*Outputs Corruption function*/
void Outputs_Corruption(int index)
{
    switch (index)
        {
        /*Outputs corruption*/
        case 1:         Vehicle_Y.Bbody[0].a()      = (real_T)index; break;
        case 2:         Vehicle_Y.Bbody[1].a()      = (real_T)index; break;
        case 3:         Vehicle_Y.Bbody[2].a()      = (real_T)index; break;
        case 4:         Vehicle_Y.Commands[0].a()   = (real_T)index; break;
        case 5:         Vehicle_Y.Commands[1].a()   = (real_T)index; break;
        case 6:         Vehicle_Y.Commands[2].a()   = (real_T)index; break;
        }
}


/*States Corruption function*/
void States_Corruption(int index)
{
    switch (index)
        {
        /*states corruption*/
        case 1:         Vehicle_DW.state[0].a()                         = (real_T)index; break;
        case 2:         Vehicle_DW.state[1].a()                         = (real_T)index; break;
        case 3:         Vehicle_DW.state[2].a()                         = (real_T)index; break;
        case 4:         Vehicle_DW.Integrator_DSTATE[0].a()             = (real_T)index; break;
        case 5:         Vehicle_DW.Integrator_DSTATE[1].a()             = (real_T)index; break;
        case 6:         Vehicle_DW.Integrator_DSTATE[2].a()             = (real_T)index; break;
        case 7:         Vehicle_DW.Integrator_DSTATE_f[0].a()           = (real_T)index; break;
        case 8:         Vehicle_DW.Integrator_DSTATE_f[1].a()           = (real_T)index; break;
        case 9:         Vehicle_DW.Integrator_DSTATE_f[2].a()           = (real_T)index; break;
        case 10:        Vehicle_DW.UD_DSTATE[0].a()                     = (real_T)index; break;
        case 11:        Vehicle_DW.UD_DSTATE[1].a()                     = (real_T)index; break;
        case 12:        Vehicle_DW.UD_DSTATE[2].a()                     = (real_T)index; break;
        case 13:        Vehicle_DW.UD_DSTATE_d[0].a()                   = (real_T)index; break;
        case 14:        Vehicle_DW.UD_DSTATE_d[1].a()                   = (real_T)index; break;
        case 15:        Vehicle_DW.UD_DSTATE_d[2].a()                   = (real_T)index; break;
        }
}


/*Constants Corruption function*/
void Consts_Corruption(int index)
{
    switch (index)
        {
        /*states corruption*/
        case 1:         Vehicle_ConstB.Abs.a()                          = (real_T)index; break;
        case 2:         Vehicle_ConstB.Abs1.a()                         = (real_T)index; break;
        case 3:         Vehicle_ConstB.Bias.a()                         = (real_T)index; break;
        case 4:         Vehicle_ConstB.Bias1.a()                        = (real_T)index; break;
        case 5:         Vehicle_ConstB.Divide.a()                       = (real_T)index; break;
        case 6:         Vehicle_ConstB.Divide1.a()                      = (real_T)index; break;
        case 7:         Vehicle_ConstB.DotProduct1.a()                  = (real_T)index; break;
        case 8:         Vehicle_ConstB.DotProduct1_c.a()                = (real_T)index; break;
        case 9:         Vehicle_ConstB.DotProduct1_g.a()                = (real_T)index; break;
        case 10:        Vehicle_ConstB.DotProduct2.a()                  = (real_T)index; break;
        case 11:        Vehicle_ConstB.DotProduct3.a()                  = (real_T)index; break;
        case 12:        Vehicle_ConstB.Gain.a()                         = (real_T)index; break;
        case 13:        Vehicle_ConstB.Gain2.a()                        = (real_T)index; break;
        case 14:        Vehicle_ConstB.Sqrt3.a()                        = (real_T)index; break;
        case 15:        Vehicle_ConstB.Sum[0].a()                       = (real_T)index; break;
        case 16:        Vehicle_ConstB.Sum[1].a()                       = (real_T)index; break;
        case 17:        Vehicle_ConstB.Sum[2].a()                       = (real_T)index; break;
        case 18:        Vehicle_ConstB.Sum_d[0].a()                     = (real_T)index; break;
        case 19:        Vehicle_ConstB.Sum_d[1].a()                     = (real_T)index; break;
        case 20:        Vehicle_ConstB.Sum_d[2].a()                     = (real_T)index; break;
        case 21:        Vehicle_ConstB.Sum_d0[0].a()                    = (real_T)index; break;
        case 22:        Vehicle_ConstB.Sum_d0[1].a()                    = (real_T)index; break;
        case 23:        Vehicle_ConstB.Sum_d0[2].a()                    = (real_T)index; break;
        case 24:        Vehicle_ConstB.Switch3[0].a()                   = (real_T)index; break;
        case 25:        Vehicle_ConstB.Switch3[1].a()                   = (real_T)index; break;
        case 26:        Vehicle_ConstB.Switch3[2].a()                   = (real_T)index; break;
        case 27:        Vehicle_ConstB.ixj.a()                          = (real_T)index; break;
        case 28:        Vehicle_ConstB.ixj_a.a()                        = (real_T)index; break;
        case 29:        Vehicle_ConstB.ixj_g.a()                        = (real_T)index; break;
        case 30:        Vehicle_ConstB.ixk.a()                          = (real_T)index; break;
        case 31:        Vehicle_ConstB.ixk_c.a()                        = (real_T)index; break;
        case 32:        Vehicle_ConstB.ixk_d.a()                        = (real_T)index; break;
        case 33:        Vehicle_ConstB.jxi.a()                          = (real_T)index; break;
```

```
        case 34:        Vehicle_ConstB.jxi_e.a()                    = (real_T)index; break;
        case 35:        Vehicle_ConstB.jxi_o.a()                    = (real_T)index; break;
        case 36:        Vehicle_ConstB.jxk.a()                      = (real_T)index; break;
        case 37:        Vehicle_ConstB.jxk_e.a()                    = (real_T)index; break;
        case 38:        Vehicle_ConstB.jxk_n.a()                    = (real_T)index; break;
        case 39:        Vehicle_ConstB.kxi.a()                      = (real_T)index; break;
        case 40:        Vehicle_ConstB.kxi_c.a()                    = (real_T)index; break;
        case 41:        Vehicle_ConstB.kxi_p.a()                    = (real_T)index; break;
        case 42:        Vehicle_ConstB.kxj.a()                      = (real_T)index; break;
        case 43:        Vehicle_ConstB.kxj_n.a()                    = (real_T)index; break;
        case 44:        Vehicle_ConstB.kxj_o.a()                    = (real_T)index; break;
        //case 45:      Vehicle_ConstB.xz.a()                       = (real_T)index; break;
        }
}


/*Block Signals Corruption function*/
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_i.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[8].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.Add.a()                       = (real_T)index; break;
            case 18:            Vehicle_B.Add2.a()                      = (real_T)index; break;
            case 19:            Vehicle_B.Add2_k.a()                        = (real_T)index; break;
            case 20:            Vehicle_B.Add_a.a()                     = (real_T)index; break;
            case 21:            Vehicle_B.Add_d.a()                     = (real_T)index; break;
            case 22:            Vehicle_B.Add_e.a()                     = (real_T)index; break;
            case 23:            Vehicle_B.Add_ek.a()                        = (real_T)index; break;
            case 24:            Vehicle_B.Add_el.a()                        = (real_T)index; break;
            case 25:            Vehicle_B.Add_f.a()                     = (real_T)index; break;
            case 26:            Vehicle_B.Add_fc.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_l.a()                     = (real_T)index; break;
            case 31:            Vehicle_B.Add_l2.a()                        = (real_T)index; break;
            case 32:            Vehicle_B.Add_m.a()                     = (real_T)index; break;
            case 33:            Vehicle_B.Add_o.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_e.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.Bias_b.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.bdot[0].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.Dipole_x[0].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 613:           Vehicle_B.Dipole_z[0].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.Divide.a()                        = (real_T)index; break;
            case 75:            Vehicle_B.Divide1.a()                       = (real_T)index; break;
            case 76:            Vehicle_B.Divide1_b.a()                     = (real_T)index; break;
            case 77:            Vehicle_B.Divide1_d.a()                     = (real_T)index; break;
            case 78:            Vehicle_B.Divide1_h.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_h.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_g.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_oi.a()              = (real_T)index; break;
        case 124:       Vehicle_B.Gain1_ox.a()              = (real_T)index; break;
        case 125:       Vehicle_B.Gain2.a()               = (real_T)index; break;
        case 126:       Vehicle_B.Gain2_a.a()               = (real_T)index; break;
        case 127:       Vehicle_B.Gain2_b.a()               = (real_T)index; break;
        case 128:       Vehicle_B.Gain2_c.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_l.a()               = (real_T)index; break;
        case 133:       Vehicle_B.Gain2_l2.a()              = (real_T)index; break;
        case 134:       Vehicle_B.Gain2_n.a()               = (real_T)index; break;
        case 135:       Vehicle_B.Gain2_o.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.Gain_o2.a()               = (real_T)index; break;
        case 139:       Vehicle_B.Gain_a.a()                = (real_T)index; break;
        case 140:       Vehicle_B.Gain_ay.a()               = (real_T)index; break;
        case 141:       Vehicle_B.Gain_b.a()                = (real_T)index; break;
        case 142:       Vehicle_B.Gain_g.a()                = (real_T)index; break;
        case 143:       Vehicle_B.Gain_i.a()                = (real_T)index; break;
        case 144:       Vehicle_B.Gain_j.a()                = (real_T)index; break;
        case 145:       Vehicle_B.Gain_k.a()                = (real_T)index; break;
        case 146:       Vehicle_B.Gain_kp.a()               = (real_T)index; break;
        case 147:       Vehicle_B.Gain_l.a()                = (real_T)index; break;
        case 148:       Vehicle_B.Gain_o.a()                = (real_T)index; break;
        case 149:       Vehicle_B.Gain_o2.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_p[0].a()           = (real_T)index; break;
        case 155:       Vehicle_B.IntegralGain_p[1].a()           = (real_T)index; break;
        case 156:       Vehicle_B.IntegralGain_p[2].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[0].a()            = (real_T)index; break;
        case 167:       Vehicle_B.is180degRot_l[1].a()            = (real_T)index; break;
        case 168:       Vehicle_B.is180degRot_l[2].a()            = (real_T)index; break;
        case 169:       Vehicle_B.ixj.a()                 = (real_T)index; break;
        case 170:       Vehicle_B.ixj_f.a()               = (real_T)index; break;
        case 171:       Vehicle_B.ixj_h.a()               = (real_T)index; break;
        case 172:       Vehicle_B.ixj_k.a()               = (real_T)index; break;
        case 173:       Vehicle_B.ixj_l.a()               = (real_T)index; break;
        case 174:       Vehicle_B.ixj_n.a()               = (real_T)index; break;
        case 175:       Vehicle_B.ixj_o.a()               = (real_T)index; break;
        case 176:       Vehicle_B.ixk.a()                 = (real_T)index; break;
        case 177:       Vehicle_B.ixk_a.a()               = (real_T)index; break;
        case 178:       Vehicle_B.ixk_b.a()               = (real_T)index; break;
        case 179:       Vehicle_B.ixk_e.a()               = (real_T)index; break;
        case 180:       Vehicle_B.ixk_g.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.MathFunction1_b.a()            = (real_T)index; break;
        case 194:       Vehicle_B.MathFunction1_m.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_i[0].a()               = (real_T)index; break;
        case 200:       Vehicle_B.Merge_i[1].a()               = (real_T)index; break;
        case 201:       Vehicle_B.Merge_i[2].a()               = (real_T)index; break;
        case 202:       Vehicle_B.Merge_i[3].a()               = (real_T)index; break;
        case 203:       Vehicle_B.Merge_il.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_ai.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_bp.a()              = (real_T)index; break;
        case 226:       Vehicle_B.Product1_bt.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_d1.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_ew.a()              = (real_T)index; break;
        case 233:       Vehicle_B.Product1_f.a()               = (real_T)index; break;
        case 234:       Vehicle_B.Product1_fh.a()              = (real_T)index; break;
        case 235:       Vehicle_B.Product1_fm.a()              = (real_T)index; break;
        case 236:       Vehicle_B.Product1_fv.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_in.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_lr.a()              = (real_T)index; break;
        case 252:       Vehicle_B.Product1_m.a()               = (real_T)index; break;
        case 253:       Vehicle_B.Product1_mu.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_bc.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 267:       Vehicle_B.Product2_e.a()               = (real_T)index; break;
        case 268:       Vehicle_B.Product2_ea.a()              = (real_T)index; break;
        case 269:       Vehicle_B.Product2_ep.a()              = (real_T)index; break;
        case 270:       Vehicle_B.Product2_eq.a()              = (real_T)index; break;
        case 271:       Vehicle_B.Product2_f.a()               = (real_T)index; break;
        case 272:       Vehicle_B.Product2_fh.a()              = (real_T)index; break;
        case 273:       Vehicle_B.Product2_fhq.a()             = (real_T)index; break;
        case 274:       Vehicle_B.Product2_g.a()               = (real_T)index; break;
        case 275:       Vehicle_B.Product2_gh.a()              = (real_T)index; break;
        case 276:       Vehicle_B.Product2_gk.a()              = (real_T)index; break;
        case 277:       Vehicle_B.Product2_i.a()               = (real_T)index; break;
        case 278:       Vehicle_B.Product2_j.a()               = (real_T)index; break;
        case 279:       Vehicle_B.Product2_k.a()               = (real_T)index; break;
        case 280:       Vehicle_B.Product2_ke.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 283:        Vehicle_B.Product2_m.a()                  = (real_T)index; break;
        case 284:        Vehicle_B.Product2_mf.a()                 = (real_T)index; break;
        case 285:        Vehicle_B.Product2_n.a()                  = (real_T)index; break;
        case 286:        Vehicle_B.Product2_nn.a()                 = (real_T)index; break;
        case 287:        Vehicle_B.Product2_no.a()                 = (real_T)index; break;
        case 288:        Vehicle_B.Product2_nx.a()                 = (real_T)index; break;
        case 289:        Vehicle_B.Product2_nz.a()                 = (real_T)index; break;
        case 290:        Vehicle_B.Product2_o.a()                  = (real_T)index; break;
        case 291:        Vehicle_B.Product2_oi.a()                 = (real_T)index; break;
        case 292:        Vehicle_B.Product2_ox.a()                 = (real_T)index; break;
        case 293:        Vehicle_B.Product2_p.a()                  = (real_T)index; break;
        case 294:        Vehicle_B.Product2_pu.a()                 = (real_T)index; break;
        case 295:        Vehicle_B.Product3.a()                  = (real_T)index; break;
        case 296:        Vehicle_B.Product3_a.a()                  = (real_T)index; break;
        case 297:        Vehicle_B.Product3_aj.a()                 = (real_T)index; break;
        case 298:        Vehicle_B.Product3_aw.a()                 = (real_T)index; break;
        case 299:        Vehicle_B.Product3_ay.a()                 = (real_T)index; break;
        case 300:        Vehicle_B.Product3_c.a()                  = (real_T)index; break;
        case 301:        Vehicle_B.Product3_c1.a()                 = (real_T)index; break;
        case 302:        Vehicle_B.Product3_c5.a()                 = (real_T)index; break;
        case 303:        Vehicle_B.Product3_cc.a()                 = (real_T)index; break;
        case 304:        Vehicle_B.Product3_cd.a()                 = (real_T)index; break;
        case 305:        Vehicle_B.Product3_ce.a()                 = (real_T)index; break;
        case 306:        Vehicle_B.Product3_d.a()                  = (real_T)index; break;
        case 307:        Vehicle_B.Product3_dh.a()                 = (real_T)index; break;
        case 308:        Vehicle_B.Product3_dz.a()                 = (real_T)index; break;
        case 309:        Vehicle_B.Product3_e.a()                  = (real_T)index; break;
        case 310:        Vehicle_B.Product3_f.a()                  = (real_T)index; break;
        case 311:        Vehicle_B.Product3_g.a()                  = (real_T)index; break;
        case 312:        Vehicle_B.Product3_gx.a()                 = (real_T)index; break;
        case 313:        Vehicle_B.Product3_h.a()                  = (real_T)index; break;
        case 314:        Vehicle_B.Product3_h5.a()                 = (real_T)index; break;
        case 315:        Vehicle_B.Product3_hf.a()                 = (real_T)index; break;
        case 316:        Vehicle_B.Product3_i.a()                  = (real_T)index; break;
        case 317:        Vehicle_B.Product3_k.a()                  = (real_T)index; break;
        case 318:        Vehicle_B.Product3_k4.a()                 = (real_T)index; break;
        case 319:        Vehicle_B.Product3_l.a()                  = (real_T)index; break;
        case 320:        Vehicle_B.Product3_l0.a()                 = (real_T)index; break;
        case 321:        Vehicle_B.Product3_ll.a()                 = (real_T)index; break;
        case 322:        Vehicle_B.Product3_lu.a()                 = (real_T)index; break;
        case 323:        Vehicle_B.Product3_lx.a()                 = (real_T)index; break;
        case 324:        Vehicle_B.Product3_lz.a()                 = (real_T)index; break;
        case 325:        Vehicle_B.Product3_lzz.a()                = (real_T)index; break;
        case 326:        Vehicle_B.Product3_n.a()                  = (real_T)index; break;
        case 327:        Vehicle_B.Product3_p.a()                  = (real_T)index; break;
        case 328:        Vehicle_B.Product3_pa.a()                 = (real_T)index; break;
        case 329:        Vehicle_B.Product3_pe.a()                 = (real_T)index; break;
        case 330:        Vehicle_B.Product3_ph.a()                 = (real_T)index; break;
        case 331:        Vehicle_B.Product3_pq.a()                 = (real_T)index; break;
        case 332:        Vehicle_B.Product3_pv.a()                 = (real_T)index; break;
        case 333:        Vehicle_B.Product4.a()                  = (real_T)index; break;
        case 334:        Vehicle_B.Product4_a.a()                  = (real_T)index; break;
        case 335:        Vehicle_B.Product4_c.a()                  = (real_T)index; break;
        case 336:        Vehicle_B.Product4_d.a()                  = (real_T)index; break;
        case 337:        Vehicle_B.Product4_g.a()                  = (real_T)index; break;
        case 338:        Vehicle_B.Product4_i.a()                  = (real_T)index; break;
        case 339:        Vehicle_B.Product4_k.a()                  = (real_T)index; break;
        case 340:        Vehicle_B.Product4_m.a()                  = (real_T)index; break;
        case 341:        Vehicle_B.Product4_o.a()                  = (real_T)index; break;
        case 342:        Vehicle_B.Product4_o5.a()                 = (real_T)index; break;
        case 343:        Vehicle_B.Product4_ok.a()                 = (real_T)index; break;
        case 344:        Vehicle_B.Product4_p.a()                  = (real_T)index; break;
        case 345:        Vehicle_B.Product4_pt.a()                 = (real_T)index; break;
        case 346:        Vehicle_B.Product5.a()                  = (real_T)index; break;
        case 347:        Vehicle_B.Product5_a.a()                  = (real_T)index; break;
        case 348:        Vehicle_B.Product5_b.a()                  = (real_T)index; break;
        case 349:        Vehicle_B.Product5_d.a()                  = (real_T)index; break;
        case 350:        Vehicle_B.Product5_g.a()                  = (real_T)index; break;
        case 351:        Vehicle_B.Product5_i.a()                  = (real_T)index; break;
        case 352:        Vehicle_B.Product5_ij.a()                 = (real_T)index; break;
        case 353:        Vehicle_B.Product5_j.a()                  = (real_T)index; break;
        case 354:        Vehicle_B.Product5_jx.a()                 = (real_T)index; break;
        case 355:        Vehicle_B.Product5_l.a()                  = (real_T)index; break;
        case 356:        Vehicle_B.Product5_n.a()                  = (real_T)index; break;
        case 357:        Vehicle_B.Product5_n2.a()                 = (real_T)index; break;
        case 358:        Vehicle_B.Product5_o.a()                  = (real_T)index; break;
        case 359:        Vehicle_B.Product6.a()                  = (real_T)index; break;
        case 360:        Vehicle_B.Product6_a.a()                  = (real_T)index; break;
        case 361:        Vehicle_B.Product6_b.a()                  = (real_T)index; break;
        case 362:        Vehicle_B.Product6_c.a()                  = (real_T)index; break;
        case 363:        Vehicle_B.Product6_d.a()                  = (real_T)index; break;
        case 364:        Vehicle_B.Product6_d5.a()                 = (real_T)index; break;
        case 365:        Vehicle_B.Product6_e.a()                  = (real_T)index; break;
        case 366:        Vehicle_B.Product6_j.a()                  = (real_T)index; break;
        case 367:        Vehicle_B.Product6_l.a()                  = (real_T)index; break;
        case 368:        Vehicle_B.Product6_ld.a()                 = (real_T)index; break;
        case 369:        Vehicle_B.Product6_o.a()                  = (real_T)index; break;
        case 370:        Vehicle_B.Product6_p.a()                  = (real_T)index; break;
        case 371:        Vehicle_B.Product7.a()                  = (real_T)index; break;
        case 372:        Vehicle_B.Product7_a.a()                  = (real_T)index; break;
        case 373:        Vehicle_B.Product7_a4.a()                 = (real_T)index; break;
        case 374:        Vehicle_B.Product7_d.a()                  = (real_T)index; break;
        case 375:        Vehicle_B.Product7_dj.a()                 = (real_T)index; break;
        case 376:        Vehicle_B.Product7_e.a()                  = (real_T)index; break;
        case 377:        Vehicle_B.Product7_ei.a()                 = (real_T)index; break;
        case 378:        Vehicle_B.Product7_em.a()                 = (real_T)index; break;
        case 379:        Vehicle_B.Product7_g.a()                  = (real_T)index; break;
        case 380:        Vehicle_B.Product7_h.a()                  = (real_T)index; break;
        case 381:        Vehicle_B.Product7_ha.a()                 = (real_T)index; break;
        case 382:        Vehicle_B.Product7_k.a()                  = (real_T)index; break;
        case 383:        Vehicle_B.Product8.a()                  = (real_T)index; break;
        case 384:        Vehicle_B.Product8_a.a()                  = (real_T)index; break;
        case 385:        Vehicle_B.Product8_c.a()                  = (real_T)index; break;
        case 386:        Vehicle_B.Product8_d.a()                  = (real_T)index; break;
```

```
        case 387:       Vehicle_B.Product8_e.a()                = (real_T)index; break;
        case 388:       Vehicle_B.Product8_f.a()                = (real_T)index; break;
        case 389:       Vehicle_B.Product8_g.a()                = (real_T)index; break;
        case 390:       Vehicle_B.Product8_j.a()                = (real_T)index; break;
        case 391:       Vehicle_B.Product8_k.a()                = (real_T)index; break;
        case 392:       Vehicle_B.Product8_m.a()                = (real_T)index; break;
        case 393:       Vehicle_B.Product8_me.a()               = (real_T)index; break;
        case 394:       Vehicle_B.Product8_p.a()                = (real_T)index; break;
        case 395:       Vehicle_B.Product_a.a()               = (real_T)index; break;
        case 396:       Vehicle_B.Product_am.a()                = (real_T)index; break;
        case 397:       Vehicle_B.Product_b.a()               = (real_T)index; break;
        case 398:       Vehicle_B.Product_bg.a()                = (real_T)index; break;
        case 399:       Vehicle_B.Product_by.a()                = (real_T)index; break;
        case 400:       Vehicle_B.Product_c[0].a()              = (real_T)index; break;
        case 401:       Vehicle_B.Product_c[1].a()              = (real_T)index; break;
        case 402:       Vehicle_B.Product_c[2].a()              = (real_T)index; break;
        case 403:       Vehicle_B.Product_c[3].a()              = (real_T)index; break;
        case 404:       Vehicle_B.Product_c[4].a()              = (real_T)index; break;
        case 405:       Vehicle_B.Product_c[5].a()              = (real_T)index; break;
        case 406:       Vehicle_B.Product_c[6].a()              = (real_T)index; break;
        case 407:       Vehicle_B.Product_c[7].a()              = (real_T)index; break;
        case 408:       Vehicle_B.Product_c[8].a()              = (real_T)index; break;
        case 409:       Vehicle_B.Product_c3.a()                = (real_T)index; break;
        case 410:       Vehicle_B.Product_cf.a()                = (real_T)index; break;
        case 411:       Vehicle_B.Product_cq.a()                = (real_T)index; break;
        case 412:       Vehicle_B.Product_cw.a()                = (real_T)index; break;
        case 413:       Vehicle_B.Product_cy[0].a()             = (real_T)index; break;
        case 414:       Vehicle_B.Product_cy[1].a()             = (real_T)index; break;
        case 415:       Vehicle_B.Product_cy[2].a()             = (real_T)index; break;
        case 416:       Vehicle_B.Product_d.a()               = (real_T)index; break;
        case 417:       Vehicle_B.Product_d0.a()                = (real_T)index; break;
        case 418:       Vehicle_B.Product_da.a()                = (real_T)index; break;
        case 419:       Vehicle_B.Product_dj.a()                = (real_T)index; break;
        case 420:       Vehicle_B.Product_e.a()               = (real_T)index; break;
        case 421:       Vehicle_B.Product_er.a()                = (real_T)index; break;
        case 422:       Vehicle_B.Product_ew.a()                = (real_T)index; break;
        case 423:       Vehicle_B.Product_ey.a()                = (real_T)index; break;
        case 424:       Vehicle_B.Product_f.a()               = (real_T)index; break;
        case 425:       Vehicle_B.Product_fv.a()                = (real_T)index; break;
        case 426:       Vehicle_B.Product_g.a()               = (real_T)index; break;
        case 427:       Vehicle_B.Product_gr.a()                = (real_T)index; break;
        case 428:       Vehicle_B.Product_h.a()               = (real_T)index; break;
        case 429:       Vehicle_B.Product_h0.a()                = (real_T)index; break;
        case 430:       Vehicle_B.Product_h1.a()                = (real_T)index; break;
        case 431:       Vehicle_B.Product_i[0].a()              = (real_T)index; break;
        case 432:       Vehicle_B.Product_i[1].a()              = (real_T)index; break;
        case 433:       Vehicle_B.Product_i[2].a()              = (real_T)index; break;
        case 434:       Vehicle_B.Product_i2.a()                = (real_T)index; break;
        case 435:       Vehicle_B.Product_if.a()                = (real_T)index; break;
        case 436:       Vehicle_B.Product_j.a()               = (real_T)index; break;
        case 437:       Vehicle_B.Product_jb.a()                = (real_T)index; break;
        case 438:       Vehicle_B.Product_jc.a()                = (real_T)index; break;
        case 439:       Vehicle_B.Product_jz.a()                = (real_T)index; break;
        case 440:       Vehicle_B.Product_k.a()               = (real_T)index; break;
        case 441:       Vehicle_B.Product_l.a()               = (real_T)index; break;
        case 442:       Vehicle_B.Product_l5.a()                = (real_T)index; break;
        case 443:       Vehicle_B.Product_m.a()               = (real_T)index; break;
        case 444:       Vehicle_B.Product_mi.a()                = (real_T)index; break;
        case 445:       Vehicle_B.Product_n.a()               = (real_T)index; break;
        case 446:       Vehicle_B.Product_o[0].a()              = (real_T)index; break;
        case 447:       Vehicle_B.Product_o[1].a()              = (real_T)index; break;
        case 448:       Vehicle_B.Product_o[2].a()              = (real_T)index; break;
        case 449:       Vehicle_B.Product_p.a()               = (real_T)index; break;
        case 450:       Vehicle_B.Product_pb.a()                = (real_T)index; break;
        case 451:       Vehicle_B.Product_pd.a()                = (real_T)index; break;
        case 452:       Vehicle_B.Product_pe.a()                = (real_T)index; break;
        case 453:       Vehicle_B.ProportionalGain[0].a()           = (real_T)index; break;
        case 454:       Vehicle_B.ProportionalGain[1].a()           = (real_T)index; break;
        case 455:       Vehicle_B.ProportionalGain[2].a()           = (real_T)index; break;
        case 456:       Vehicle_B.ProportionalGain_h[0].a()         = (real_T)index; break;
        case 457:       Vehicle_B.ProportionalGain_h[1].a()         = (real_T)index; break;
        case 458:       Vehicle_B.ProportionalGain_h[2].a()         = (real_T)index; break;

        case 459:       Vehicle_B.Sqrt3.a()                 = (real_T)index; break;
        case 460:       Vehicle_B.Sqrt3_j.a()                 = (real_T)index; break;
        case 461:       Vehicle_B.Sqrt3_l.a()                 = (real_T)index; break;
        case 462:       Vehicle_B.Sqrt3_m.a()                 = (real_T)index; break;
        case 463:       Vehicle_B.Sqrt3_mq.a()                = (real_T)index; break;
        case 464:       Vehicle_B.sqrt_a.a()                  = (real_T)index; break;
        case 465:       Vehicle_B.sqrt_g.a()                  = (real_T)index; break;
        case 466:       Vehicle_B.sqrt_h.a()                  = (real_T)index; break;
        case 467:       Vehicle_B.sqrt_i.a()                  = (real_T)index; break;
        case 468:       Vehicle_B.sqrt_j.a()                  = (real_T)index; break;
        case 469:       Vehicle_B.sqrt_k.a()                  = (real_T)index; break;
        case 470:       Vehicle_B.sqrt_l.a()                  = (real_T)index; break;
        case 471:       Vehicle_B.sqrt_m.a()                  = (real_T)index; break;
        case 472:       Vehicle_B.sqrt_mo.a()                 = (real_T)index; break;
        case 473:       Vehicle_B.sqrt_mv.a()                 = (real_T)index; break;
        case 474:       Vehicle_B.sqrt_mw.a()                 = (real_T)index; break;
        case 475:       Vehicle_B.sqrt_n.a()                  = (real_T)index; break;

        case 476:       Vehicle_B.Sum[0].a()                  = (real_T)index; break;
        case 477:       Vehicle_B.Sum[1].a()                  = (real_T)index; break;
        case 478:       Vehicle_B.Sum[2].a()                  = (real_T)index; break;
        case 479:       Vehicle_B.Sum1.a()                  = (real_T)index; break;
        case 480:       Vehicle_B.Sum1_a.a()                  = (real_T)index; break;
        case 481:       Vehicle_B.Sum1_b.a()                  = (real_T)index; break;
        case 482:       Vehicle_B.Sum1_bu.a()                 = (real_T)index; break;
        case 483:       Vehicle_B.Sum1_d.a()                  = (real_T)index; break;
        case 484:       Vehicle_B.Sum1_d1.a()                 = (real_T)index; break;
        case 485:       Vehicle_B.Sum1_h.a()                  = (real_T)index; break;
        case 486:       Vehicle_B.Sum1_hk.a()                 = (real_T)index; break;
        case 487:       Vehicle_B.Sum1_k.a()                  = (real_T)index; break;
        case 488:       Vehicle_B.Sum1_l.a()                  = (real_T)index; break;
```

```
            case 489:        Vehicle_B.Sum1_m.a()                 = (real_T)index; break;
            case 490:        Vehicle_B.Sum1_n.a()                 = (real_T)index; break;
            case 491:        Vehicle_B.Sum2.a()                 = (real_T)index; break;
            case 492:        Vehicle_B.Sum2_a.a()                 = (real_T)index; break;
            case 493:        Vehicle_B.Sum2_g.a()                 = (real_T)index; break;
            case 494:        Vehicle_B.Sum2_i.a()                 = (real_T)index; break;
            case 495:        Vehicle_B.Sum2_k.a()                 = (real_T)index; break;
            case 496:        Vehicle_B.Sum2_l.a()                 = (real_T)index; break;
            case 497:        Vehicle_B.Sum2_la.a()                = (real_T)index; break;
            case 498:        Vehicle_B.Sum2_m.a()                 = (real_T)index; break;
            case 499:        Vehicle_B.Sum2_mr.a()                = (real_T)index; break;
            case 500:        Vehicle_B.Sum2_o.a()                 = (real_T)index; break;
            case 501:        Vehicle_B.Sum2_p.a()                 = (real_T)index; break;
            case 502:        Vehicle_B.Sum2_p3.a()                = (real_T)index; break;
            case 503:        Vehicle_B.Sum3.a()                 = (real_T)index; break;
            case 504:        Vehicle_B.Sum3_b.a()                 = (real_T)index; break;
            case 505:        Vehicle_B.Sum3_b5.a()                = (real_T)index; break;
            case 506:        Vehicle_B.Sum3_bi.a()                = (real_T)index; break;
            case 507:        Vehicle_B.Sum3_d.a()                 = (real_T)index; break;
            case 508:        Vehicle_B.Sum3_e.a()                 = (real_T)index; break;
            case 509:        Vehicle_B.Sum3_h.a()                 = (real_T)index; break;
            case 510:        Vehicle_B.Sum3_j.a()                 = (real_T)index; break;
            case 511:        Vehicle_B.Sum3_jh.a()                = (real_T)index; break;
            case 512:        Vehicle_B.Sum3_l.a()                 = (real_T)index; break;
            case 513:        Vehicle_B.Sum3_o.a()                 = (real_T)index; break;
            case 514:        Vehicle_B.Sum3_p.a()                 = (real_T)index; break;
            case 515:        Vehicle_B.Sum_b[0].a()               = (real_T)index; break;
            case 516:        Vehicle_B.Sum_b[1].a()               = (real_T)index; break;
            case 517:        Vehicle_B.Sum_b[2].a()               = (real_T)index; break;
            case 518:        Vehicle_B.Sum_c.a()                = (real_T)index; break;
            case 519:        Vehicle_B.Sum_d.a()                = (real_T)index; break;
            case 520:        Vehicle_B.Sum_d5.a()                 = (real_T)index; break;
            case 521:        Vehicle_B.Sum_e.a()                = (real_T)index; break;
            case 523:        Vehicle_B.Sum_f.a()                = (real_T)index; break;
            case 524:        Vehicle_B.Sum_ff[0].a()              = (real_T)index; break;
            case 525:        Vehicle_B.Sum_ff[1].a()              = (real_T)index; break;
            case 526:        Vehicle_B.Sum_ff[2].a()              = (real_T)index; break;
            case 527:        Vehicle_B.Sum_g.a()                = (real_T)index; break;
            case 528:        Vehicle_B.Sum_g5.a()                 = (real_T)index; break;
            case 529:        Vehicle_B.Sum_gh.a()                 = (real_T)index; break;
            case 530:        Vehicle_B.Sum_gt.a()                 = (real_T)index; break;
            case 531:        Vehicle_B.Sum_gz.a()                 = (real_T)index; break;
            case 532:        Vehicle_B.Sum_i.a()                = (real_T)index; break;
            case 533:        Vehicle_B.Sum_iw.a()                 = (real_T)index; break;
            case 534:        Vehicle_B.Sum_j.a()                = (real_T)index; break;
            case 535:        Vehicle_B.Sum_k.a()                = (real_T)index; break;
            case 536:        Vehicle_B.Sum_kb.a()                 = (real_T)index; break;
            case 537:        Vehicle_B.Sum_kc.a()                 = (real_T)index; break;
            case 538:        Vehicle_B.Sum_ki.a()                 = (real_T)index; break;
            case 539:        Vehicle_B.Sum_kk.a()                 = (real_T)index; break;
            case 540:        Vehicle_B.Sum_l[0].a()               = (real_T)index; break;
            case 541:        Vehicle_B.Sum_l[1].a()               = (real_T)index; break;
            case 542:        Vehicle_B.Sum_l[2].a()               = (real_T)index; break;
            case 543:        Vehicle_B.Sum_lm.a()                 = (real_T)index; break;
            case 544:        Vehicle_B.Sum_ln.a()                 = (real_T)index; break;
            case 545:        Vehicle_B.Sum_m[0].a()               = (real_T)index; break;
            case 546:        Vehicle_B.Sum_m[1].a()               = (real_T)index; break;
            case 547:        Vehicle_B.Sum_m[2].a()               = (real_T)index; break;
            case 548:        Vehicle_B.Sum_n.a()                = (real_T)index; break;
            case 549:        Vehicle_B.Sum_n1.a()                 = (real_T)index; break;
            case 550:        Vehicle_B.Sum_n3.a()                 = (real_T)index; break;
            case 551:        Vehicle_B.Sum_nx.a()                 = (real_T)index; break;
            case 552:        Vehicle_B.Sum_o.a()                = (real_T)index; break;
            case 553:        Vehicle_B.Sum_oa.a()                 = (real_T)index; break;
            case 554:        Vehicle_B.Sum_oq.a()                 = (real_T)index; break;
            case 555:        Vehicle_B.Sum_oz.a()                 = (real_T)index; break;
            case 556:        Vehicle_B.Sum_p.a()                = (real_T)index; break;
            case 557:        Vehicle_B.Sum_pb[0].a()              = (real_T)index; break;
            case 558:        Vehicle_B.Sum_pb[1].a()              = (real_T)index; break;
            case 559:        Vehicle_B.Sum_pb[2].a()              = (real_T)index; break;
            case 560:        Vehicle_B.Sum_pd[0].a()              = (real_T)index; break;
            case 561:        Vehicle_B.Sum_pd[1].a()              = (real_T)index; break;
            case 562:        Vehicle_B.Sum_pd[2].a()              = (real_T)index; break;
            case 563:        Vehicle_B.Sum_pw.a()                 = (real_T)index; break;
            case 564:        Vehicle_B.SumofElements_c.a()            = (real_T)index; break;
            case 565:        Vehicle_B.SumofElements_l.a()            = (real_T)index; break;
            case 566:        Vehicle_B.Switch[0].a()              = (real_T)index; break;
            case 567:        Vehicle_B.Switch[1].a()              = (real_T)index; break;
            case 568:        Vehicle_B.Switch3[0].a()               = (real_T)index; break;
            case 569:        Vehicle_B.Switch3[1].a()               = (real_T)index; break;
            case 570:        Vehicle_B.Switch3[2].a()               = (real_T)index; break;
            case 571:        Vehicle_B.Switch_b[0].a()              = (real_T)index; break;
            case 572:        Vehicle_B.Switch_b[1].a()              = (real_T)index; break;
            case 573:        Vehicle_B.Switch_c[0].a()              = (real_T)index; break;
            case 574:        Vehicle_B.Switch_c[1].a()              = (real_T)index; break;
            case 575:        Vehicle_B.Switch_c[2].a()              = (real_T)index; break;
            case 576:        Vehicle_B.Switch_c[3].a()              = (real_T)index; break;
            case 577:        Vehicle_B.Switch_f[0].a()              = (real_T)index; break;
            case 578:        Vehicle_B.Switch_f[1].a()              = (real_T)index; break;
            case 579:        Vehicle_B.Switch_f[0].a()              = (real_T)index; break;

            case 580:        Vehicle_B.TmpSignalConversionAtDotProdu_d[0].a()              = (real_T)index; break
    ;
            case 581:        Vehicle_B.TmpSignalConversionAtDotProdu_d[1].a()              = (real_T)index; break
    ;
            case 582:        Vehicle_B.TmpSignalConversionAtDotProdu_d[2].a()              = (real_T)index; break
    ;
            case 583:        Vehicle_B.TmpSignalConversionAtDotProdu_j[0].a()              = (real_T)index; break
    ;
            case 584:        Vehicle_B.TmpSignalConversionAtDotProdu_j[1].a()              = (real_T)index; break
    ;
            case 585:        Vehicle_B.TmpSignalConversionAtDotProdu_j[2].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.UD[0].a()                       = (real_T)index; break;
            case 596:         Vehicle_B.UD[1].a()                       = (real_T)index; break;
            case 597:         Vehicle_B.UD[2].a()                       = (real_T)index; break;
            case 598:         Vehicle_B.UD_l[0].a()                       = (real_T)index; break;
            case 599:         Vehicle_B.UD_l[1].a()                       = (real_T)index; break;
            case 600:         Vehicle_B.UD_l[2].a()                       = (real_T)index; break;
            case 601:         Vehicle_B.UnaryMinus.a()                       = (real_T)index; break;
            case 602:         Vehicle_B.UnaryMinus1.a()                       = (real_T)index; break;
            case 603:         Vehicle_B.UnaryMinus2.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

[1] Leonardo reyneri, Alejandro cases, Yolanda morilla, Sergio Asensi, and Antonio Martinez. «A compact model to evaluate the effects of high level C++ code hardening in radiation environments». In: 1 (Apr. 2019) (cit. on p. 2).

[2] Ibrahim Kök. «Comparison and Analysis of Attitude Control Systems of a Satellite Using Reaction Wheel Actuators». In: (Oct. 2012) (cit. on pp. 4, 6, 7, 28, 29).

[3] Muhammad Yasir. «Development and Implementation of the attitude control algorithms for the micro-satellite Flying Laptop». In: (Mar. 2010) (cit. on pp. 4, 5, 8, 32).

[4] «https://i.stack.imgur.com/A33El.jpg». In: () (cit. on p. 6).

[5] «https://www.allaboutcircuits.com/technical-articles/dont-get-lost-in-deep-space-understanding-quaternions/#:~:text=Quaternions/20are/20an/20alternate/20way,not/20suffer/20from/20gimbal/20lock.». In: (2017) (cit. on p. 8).

[6] Catanzaro Alessandro Rocco. «Design and simulation analysis of Attitude and Determination Control System for the CubeSat». In: () (cit. on p. 9).

[7] Nnamdi N. Jibiri Victor U. J. Nwankwo and Michael T. Kio. «The Impact of Space Radiation Environment on Satellites Operation in Near-Earth Space». In: (June 2020) (cit. on pp. 9–11).

[8] «https://it.mathworks.com/matlabcentral/fileexchange/70030-aerospace-blockset-cubesat-simulation-library». In: (Sept. 2020) (cit. on p. 13).

[9] «https://www.mathworks.com/help/releases/R2020b/aeroblks/sphericalharmonicgravitymodel.html». In: (2010) (cit. on p. 15).

[10] Paweł Zagórski. «Modeling disturbances influencing an Earth-orbiting satellite». In: (May 2012) (cit. on p. 17).