Project: Generic AOCS/GNC Techniques & Design Framework for FDIR

Title: GAFE User's Manual

Doc. No.: GAFE-UM-D7.5b
Issue: 2.1
Date: 17.08.2018

	Name	Institution
Author(s):	Marc Hirth	Astos Solutions GmbH
	Haifeng Su	Astos Solutions GmbH
	Domenico Reggio	Airbus Defence & Space
	Patrick Bergner	Airbus Defence & Space

DISTRIBUTION LIST

Quantity	Type	Name	Company/Department
1	PDF	Alvaro Martinez Barrio	European Space Agency, ESTEC
1	PDF	Marcel Verhoef	European Space Agency, ESTEC
1	PDF	Study Team	Airbus DS GmbH, Astos Solutions GmbH, iFR Universität Stuttgart

CHANGE RECORD

Issue	Rev.	Date	Pages/Section	Changes
1	0	29.04.2018	All	Initial Issue
2	0	12.06.2018	4.6.2, 4.6.4.2, 1.3, 3.7	Updated according to feedback of “Final Review”
2	1	17.08.2018	2.3	Updated installation procedure

TABLE OF CONTENTS

1 Introduction

1.1 Scope of the Document

This is the User's Manual of the GAFE Framework, which consists of two tools: the GAFE Structural Analysis and the GAFE Simulator.

The objective of this manual is to provide the user all information necessary to start working with the tools, but also to understand the concepts behind them. This eases their usage, but is also a key element for extension of the tools.

Several elements implemented in the GAFE Structural Analysis and the GAFE Simulator have strong relations to the GAFE Methodology. We consider it very helpful for understanding to be familiar with the methodology before reading this User's Manual.

For a quick start we recommend the GAFE Demo Description (referenced below). It summarizes very briefly the steps needed to get the two GAFE tools running.

1.2 Applicable Documents and Reference Documents

Reference Documents

Any other document, published or not, that is reference in this document is listed below.

[RD-1] GAFE Website, http://gafe.estec.esa.int

[RD-2] GAFE Methodology, GAFE-UM-D7.5a, Issue 1.0

[RD-3] GAFE Demo Description, GAFE-RP-A1, Issue 3.0
Located in framework under: Gafe\Documentation\

[RD-4] User Manual of Fault Diagnosis Toolbox (2017-09-08), faultdiagnosistoolbox.github.io/

[RD-5] FDIR Module, Technical Documentation & User Guide, Issue 1.0.
Located in framework under: Gafe\Simulator\ext\FdirModule\doc\general\UserGuide\

1.3 Protected Functions

Some functionality is only available as protected functions. However, the full functionality including the description is assured.

The following sublibraries and functions are protected:

- functions which support testing of new AOCS Algorithmic Components (AutoDoc)
- signal manipulation functions (SigMa)
- some mathematical support functions (SFLIB)
- functions which allow manipulation and plotting of structures (Fields)
- selected AOCS Algorithmic Components which are based on EarthCARE algorithms

2 GAFE – Framework

2.1 Hardware & Software Prerequisites

GAFE is developed and successfully tested with the following Hardware and Software

- PC / Laptop (16GB RAM)
- Windows 7
- Monitor with Full HD Resolution (for the plots)
- Matlab / Simulink R2016b

2.2 Separation of GAFE Framework & GAFE Projects

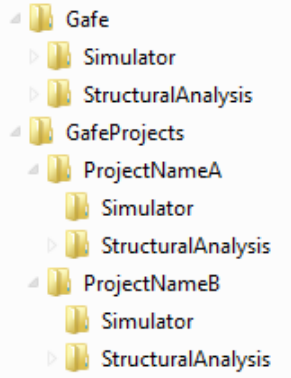
GAFE is separated into the GAFE Library (“Gafe”, incl. all library components, documentation and support functions) and the GAFE Projects (“GafeProjects”).

Both tools of the GAFE, the GAFE Simulator and the GAFE Structural Analysis, are designed to support this separation. The basic concept is: project specific files are included on top of the MATLAB path and library files further below. If the project contains a file of the same name as one in the library, the project version of this file is taken.

The recommended top-level folder structure for the GAFE Framework and e.g. two projects (A and B) would be:

```
Gafe\
Gafe\Simulator\
Gafe\StructuralAnalysis\

GafeProjects\ProjectA\
GafeProjects\ProjectA\Simulator\
GafeProjects\ProjectA\StructuralAnalysis\

GafeProjects\ProjectB\
GafeProjects\ProjectB\Simulator\
GafeProjects\ProjectB\StructuralAnalysis\
```

or as tree-view:



This guiding principle allows to start a project with only the bare minimum of user information (the really project specific configuration) and leave the rest initially untouched. When the project is detailed further, other files can be copied from the library to the project and be modified there according to the project needs.

2.3 Installation & General Usage

GAFE does not need to be installed; instead it can just be unzipped and copied to any desired folder.

In a second step the Fault Diagnosis toolbox in its version "FaultDiagnosisToolbox_2017-09-08" should be downloaded from https://faultdiagnosistoolbox.github.io/download/ and unzipped in the GAFE subfolder `"StructuralAnalysis/ext"`, such that subfolder `"StructuralAnalysis/ext/FaultDiagnosisToolbox_2017-09-08/src"` exists.

In order to use GAFE, first the required paths have to be set in MATLAB. For that purpose, simply run the file `"gafeStartup.m"` in the GAFE main folder.

When working on a project, it is additionally necessary to set the paths also for this project – by similarly executing the script `"gafeProjectStartup.m"` in the individual project folder. It is recommended to execute this script each time before running a simulation or doing a structural analysis for a project (to make sure that any created folders are actually on the path).

Nominally, GAFE serves as library and all user modifications (parameterization, modification of models, etc.) shall be realized within a project. Please refer to section 4.4.2 on how to create a GAFE project.

When switching between projects, it is always necessary to run `"gafeStartup.m"` followed by `"gafeProjectStartup.m"` of the current project to ensure that no files are shadowed by any other project.

3 GAFE – Structural Analysis

The objective of this section is to show the user how to use the GAFE Structural Analysis Tool. This will be done by giving a brief overview of the

- Purpose of the Structural Analysis Tool
- Tool organisation (folder structure)
- Creating a new project
- Project configuration
- Running the Analysis
- Analysis Outputs
- Overview of existing model
- Tool extension (adding equipment models)

3.1 Purpose of Structural Analysis Tool

The GAFE Structural Analysis Tool is based, as the name says, on a mathematical method called “structural analysis”. This method focusses on the relations between known and unknown “states” of a system and allows to identify if it is possible to detect and/or isolate (unambiguously identify) faults in predefined elements of the system. A general introduction into the method itself is given in[RD-2], Appendix A.

The GAFE Methodology is subdivided in seven tasks (see [RD-2], Section 3.1). The purpose of the GAFE Structural Analysis Tool is to assist a user in applying the methodology during task 2 (Extension of Nominal Equipment Set) and parts of task 3 (Definition & Implementation of FDIR Concept).

The extension of the Nominal Equipment Set (NES) in task 2 has the goal to make this set compliant with the required fault diagnosis capabilities (i.e. fault detection and potentially isolation for all considered faults) and the requested failure tolerance requirement (fault recovery) for all AOCS modes. In general this task consists of adding or activating additional AOCS units and/or analytic redundancy relations. The outcome of this task is the Extended Equipment Set for Failure Detection (, Isolation) and Recovery (EES-FDR/FDIR), as explained in [RD-2], section 4.2.

The parts of the Definition & Implementation of FDIR Concept (task 3, see [RD-2] section 4.3), assisted by the tool are the determination of the fault detectability and isolability information and of the fault signatures and residual structures required for the implementation of the analytic redundancy relations for all AOCS modes (see [RD-2], section 4.3.2) .

Acknowledgement: The GAFE Structural Analysis Tool is built around a toolbox developed by Erik Frisk and Mattias Krysander from of Linköping University, Sweden. This Fault Diagnosis Toolbox (see [RD-4]) provides an easy to use MATLAB implementation of the main functions required for a “structural analysis”.

3.1.1 Workflow of the Tool

In contrast to the manual approach of adding or activating single additional AOCS units and/or analytic redundancy relations in a sequential way until the required fault diagnosis capability (FD or FDI) for all AOCS modes is achieved, the GAFE Structural Analysis Tool performs this task in a different way.

Based on project specific inputs (general and for each AOCS mode) it generates a so-called evaluation table of possible AOCS hardware sets and then systematically discards all sets which are not compatible with constraints defined by the user, e.g. because a set

- does not contain enough AOCS units for all AOCS modes,
- does not allow to replace any failed unit,
- does contain forbidden equipment.

Afterwards, the remaining HW sets are investigated for

- the fault diagnosis capability they provide together with the allowed analytic redundancy relations (i.e. detectability and isolability of considered faults)
- their costs in terms of price, weight, power consumption and CPU load (e.g. for sensor processing).

The best solution is determined and then further investigated to determine the required analytic redundancy relations, the necessary residuals and the fault signatures to be implemented and activated per AOCS mode as part of the FDIR onboard the spacecraft.

The fault diagnosis capability of AOCS hardware sets is determined using the functions implemented in the Fault Diagnosis Toolbox (see [RD-4]), which is integrated in and used by this tool.

3.1.2 Assumptions

The GAFE structural analysis tool has been developed considering the following assumptions:

- Fault detection and isolation is restricted to one fault per time.
- Any AOCS unit of the equipment set can by faulty.
- If an AOCS unit can become unavailable during normal operation (e.g. a star tracker can be blinded) this possibility must be defined using the concept of the unit unavailability vector (UUV), see [RD-2] section 2.2.2).
- Only one UUV is considered to be active per time.
- Analytical models are in general considered to be free of faults (V&V is completed before flight).
- Analytical models are never unavailable as long as their inputs are not unavailable.
- The fault diagnosis capability required for an AOCS mode must hold for all UUVs defined for this mode individually.

- All analytic redundancy relations allowed by the user are considered usable for FDI purposes during the investigation of the fault diagnosis capability of a hardware set.

3.2 Tool Organization

This section shows how the tool is organized from a folder point of view, allowing the user to navigate through the tool and find the correct files.

The general folder organization of the GAFE library and the GAFE projects, as well as the guiding principle behind, is discussed in Section 2.2. The main organization scheme is repeated in Figure 3-1 below for convenience.

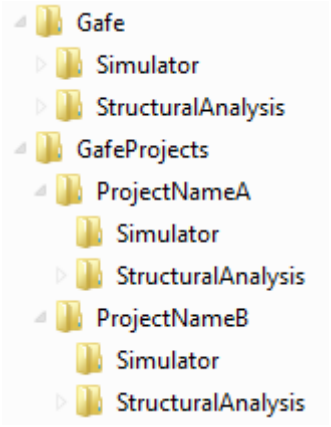


Figure 3-1: Top level folders structure of GAFE

The main folder structure of the GAFE Structural Analysis is show in Figure 3-2. The purpose of the different folders is:

- `bin:` all tool generated data is stored here
  - Content: analysis results and log files (after running an analysis)
- `data:` configuration files
  - Content: configuration of project, analysis, equipment models and analytic redundancy relation models
- `doc:` documentation
  - Content: documentation files for functions, function unit tests
- `ext:` external inputs
  - Content: Fault Diagnosis Toolbox and some of its reference documents.
- `src:` source code of the tool
  - `functions:` main functions
  - `models:` structural models
    - Content: equipment models (sensors, actuators), analytic redundancy models
  - `supportFcns:` support functions
    - Content: post processing functions, generic functions, little helpers

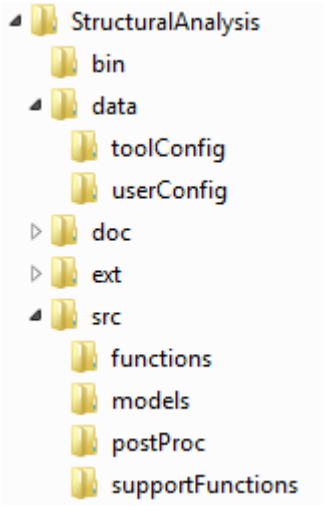


Figure 3-2 Top level folders structure of GAFE Structural Analysis.

3.3 Creating a New Project

Give that the installation instructions from Section 2.3 have been already performed; a new project can be created with following steps:

- Open MATLAB 2016b
- Navigate to the folder “Gafe”
- Start-up GAFE by executing the function

```
> gafeStartup()
```

  This command mainly sets the paths to required functions.

- Navigate to the folder “GafeProjects”
- Execute the following command to create a new project with the given project name in the GafeProjects folder:

```
> createProject('ProjectName')
```

  Info: This command creates all required folders, copies the configuration files (see Section 3.4) from the GAFE library to the new project and creates a gafeProjectStartup() file.

- Execute the following command to make the project just created the current project to work on:

```
> gafeProjectStartup()
```

3.4 Project Configuration

This section shows how the user can configure the Structural Analysis Tool in terms of project specific data and analysis settings.

There are four main configuration files: configProject, configAnalysis, configEquipment, and configModels. In the following subsections the content of each of them is explained in detail.

3.4.1 Configuration of Project

The basic configuration of the project to analysis takes place in `configProject.m`. This includes the definition of the allowed equipment, the allowed analytic redundancy models (both mode independent), and all kind of AOCS mode specific information.

3.4.1.1 AOCS Mode Independent Configuration

► *Cost Weighting Factor:*

Defines the weighting of the four cost factors for AOCS equipment and analytic redundancy models: computational cost, weight (mass), power consumption and price).

```
% Definition of cost weighting factor
% [computationalCost, unitWeight, powerConsumption, priceOfUnit]
costWeightingFactor = [0.25, 0.40, 0.20, 0.15];
```

► *Allowed Equipment:*

Defines which AOCS equipment (sensors and actuators) is generally allowed to be considered in the analysis.

```
% Allowed Equipment
% List of equipment allowed to be used in the project
% Options: 0 = exclude, 1 = include
allowedEquipment = {'acc'    1;   % Accelerometer
                    'cam'    1;   % Camera
                    'cess'   1;   % Coarse Earth Sun Sensor
                    'es'     0;   % Earth Sensor
                    'gnsr'   1;   % GNS Receiver
                    'lidar'  0;   % LIDAR
                    'mag'    1;   % Magnetometer (3-axes)
                    'mtq'    0;   % Magnetorquer (3-axes)
                    'obtAbs' 0;   % Absolute OBT
                    'rcs'    0;   % Reaction Control System
                    'rmu'    1;   % Rate Measurement Units (3-axes)
                    'rw'     1;   % Reaction Wheel
                    'sadm'   0;   % Solar Array Drive
                    'ss'     0;   % Sun Sensor
                    'str'    1};  % Star Tracker
```

► *Allowed (Analytic Redundancy) Models:*

Defines which analytic redundancy models are generally allowed to be considered in the analysis.

```
% Allowed Models
% List of models (ARRs) allowed to be used in the project
% Options: 0 = exclude, 1 = include
allowedModels = {'eomTrans'        1;   % Equations of Motion Transl.
                 'eomRot'          1;   % Equations of Motion Rotation
                 'earthKin'        1;   % Earth Kinematics
                 'earthMagField'   1;   % Earth Magnetic Field
                 'earthAtmDens'    1;   % Earth Atmospheric Density
                 'earthDir'        1;   % Earth Direction
                 'sunDir'          1;   % Sun Direction
                 'attTrf'          1;   % Attitude Transformations
                 'posTrf'          1;   % Position Transformation
                 'velTrf'          1;   % Velocity Transformation
                 'timeAbsRel'      1;   % Time Absolute/Relative
                 'earthGrav'       1;   % Earth Gravity
                 'nonGravAcc'      1;   % Non-Gravitational Acc.
                 'relPosDir'       1};  % Rel. Pos. Dir. Transf.
```

► *Minimum HW required:*

Defines the minimum number of units per AOCS equipment to be considered. The analysis will not consider solutions which do not fulfill this.

```
% Minimum HW Required
% Default is zero units per equipment
minHwRequired = {'cess'   0;   % Coarse Earth Sun Sensor
                 'cam'    0;   % Camera
                 'es'     0;   % Earth Sensor
                 'gnsr'   0;   % GNS Receiver
                 'lidar'  0;   % LIDAR
                 'mag'    0;   % Magnetometer (3-axes)
                 'mtq'    0;   % Magnetorquer (3-axes)
                 'obtAbs' 0;   % Absolute OBT
                 'rcs'    0;   % Reaction Control System
                 'rmu'    0;   % Rate Measurement Units (3-axes)
                 'rw'     0;   % Reaction Wheel
                 'sadm'   0;   % Solar Array Drive
                 'ss'     0;   % Sun Sensor
                 'str'    0};  % Star Tracker
```

► *Maximum HW possible:*

Defines the maximum number of units per AOCS equipment to be considered. The analysis will not consider solutions which do not fulfill this.

Remark: As default an algorithm is used to compute the maximum HW possible automatically. In case the user sets any value in 'maxHwPossible' different from zeros, the maximum HW for all equipment types will be entirely defined by 'maxHwPossible'.

```
% Maximum HW Possible
maxHwPossible = {'cess'    0;   % Coarse Earth Sun Sensor
                 'cam'     0;   % Camera
                 'es'      0;   % Earth Sensor
                 'gnsr'    0;   % GNS Receiver
                 'lidar'   0;   % LIDAR
                 'mag'     0;   % Magnetometer (3-axes)
                 'mtq'     0;   % Magnetorquer (3-axes)
                 'obtAbs'  0;   % Absolute OBT
                 'rcs'     0;   % Reaction Control System
                 'rmu'     0;   % Rate Measurement Units (3-axes)
                 'rw'      0;   % Reaction Wheel
                 'sadm'    0;   % Solar Array Drive
                 'ss'      0;   % Sun Sensor
                 'str'     0};  % Star Tracker
```

► *Extended Equipment Set for Fault Recovery (EES-FR):*

Defines the EES-FR (see definition in [RD-1], Extension of Nominal Equipment Set (Task 2), Step 1). The analysis will not consider solutions which do not fulfill this.

Remark: As default an algorithm is used to compute the EES-FR automatically (total NES + 1). In case the user sets any value in “eesFr” different from zeros, the “eesFr” for all equipment types will be entirely defined by “eesFr”.

```
% Extended Equipment Set for Fault Recovery (EES-FR)
eesFr = {'cess'    0;   % Coarse Earth Sun Sensor
         'cam'     0;   % Camera
         'es'      0;   % Earth Sensor
         'gnsr'    0;   % GNS Receiver
         'lidar'   0;   % LIDAR
         'mag'     0;   % Magnetometer (3-axes)
         'mtq'     0;   % Magnetorquer (3-axes)
         'obtAbs'  0;   % Absolute OBT
         'obtRel'  0;   % Relative OBT
         'rcs'     0;   % Reaction Control System
         'rmu'     0;   % Rate Measurement Units (3-axes)
         'rw'      0;   % Reaction Wheel
         'sadm'    0;   % Solar Array Drive
         'ss'      0;   % Sun Sensor
         'str'     0};  % Star Tracker
```

3.4.1.2 AOCS Mode Specific Configuration

The following configuration items have to be defined for each AOCS Mode of the project separately.

► *Mode ID:*

A continuous linear index of the n AOCS modes, numbering them from 1…n.

```
Mode ID
modeId = 1;
```

► *Mode Name:*

The name of the mode to defined.

```
Mode Name
modesConfig(modeId, 1).name = 'AocsModeA';
```

► *Nominal Equipment Set (NES):*

Defines the NES (see definition in [RD-1], section "Nominal Equipment Set")

```
Nominal Equipment Set
modesConfig(modeId, 1).nes = {'cess'   0;   % Coarse Earth Sun Sensor
                              'cam'    1;   % Camera
                              'es'     0;   % Earth Sensor
                              'gnsr'   1;   % GNS Receiver
                              'lidar'  1;   % LIDAR
                              'mag'    0;   % Magnetometer (3-axes)
                              'mtq'    0;   % Magnetorquer (3-axes)
                              'obtAbs' 0;   % Absolute OBT
                              'rcs'    1;   % Reaction Contr. Sys.
                              'rmu'    1;   % Rate Meas. Unit
                              'rw'     0;   % Reaction Wheel
                              'sadm'   0;   % Solar Array Drive
                              'ss'     1;   % Sun Sensor
                              'str'    1};  % Star Tracker
```

► *Forbidden Equipment:*

If required, this item can be used to define which type of the allowed equipment (as defined in the mode independent configuration) is forbidden to be used in the currently defined AOCS mode. This feature allows e.g. to exclude GPS or STR from an AOCS safe mode.

```
% Forbidden Equipment
modesConfig(modeId, 1).forbEq  = {'str'};
```

► *Forbidden Models:*

If required, this item can be used to define which type of the allowed models (analytic redundancy relation, ARR; as defined in the mode independent configuration) is forbidden to be used in the currently defined AOCS mode. This feature allows e.g. to exclude GPS or STR based models from an AOCS safe mode.

***Remark:** If the corresponding equipment to feed the models in not allowed (or forbidden), then the model would anyhow not show up in any solution. Nevertheless it would increase the duration of the analysis.*

```
% Forbidden Equipment
modesConfig(modeId, 1).forbArr = {'nonGravAcc'};
```

► *Required Fault Diagnosis Capability:*

The possible options for each AOCS mode are: "Fault Detection (FD)" and "Fault Detection and Isolation (FDI)". See definition in [RD-1], section "Extended Equipment Set for Failure Detection, Isolation & Recovery (Step 3)".

```
% Required Fault Diagnosis Capability
modesConfig(modeId, 1).reqCap  = 'FDI';
```

► *Unit Unavailability Vector (UUV):*

The unit unavailability vector can be used to define which units or combinations of units are expected to be temporarily unavailable in the currently defined AOCS mode. Each AOCS mode can have anything from no to several UUVs. For the exact definition and idea of the UUV please refer to [RD-1], section "Nominal Equipment Set", subsection "Measurement Configurations".

```
% Required Fault Diagnosis Capability
modesConfig(modeId, 1).uuv = {{'str', 1}
                              {'es',  2}}
```

3.4.2 Configuration of Analysis

The configuration of the analysis takes place in `configAnalysis.m`. The following settings are possible:

► *MSO Approximation Method:*

An option for the core functions of the used Fault Diagnosis Toolbox (see [RD-4]), which, if set, allows some approximation in the search for the best set of residuals to be implemented. For the use in the GAFE Structural Analysis it is recommended to use the exact solution.

```
% Use approximation method or compute exact minimal residual set.
% Options: 0 = exact
%          1 = approximation
useMsoApproximation = 0;
```

► *Plot Cost Overview:*

Flag to enable the plot of the cost overview over all solutions.

```
% Plot cost overview at end of analysis
% Options: 0 = no
%          1 = yes
plotCostOverview = 0;
```

► *Show Analysis Results Table*

Flag to enable the command line output (+log file) of the analysis results table.

```
% Show table of analysis results
% Options: 0 = no
%          1 = yes
showResultsTable = 1;
```

3.4.3 Configuration of Equipment

The configuration of the equipment models takes place in `configEquipment.m`. The following settings are possible:

► *Equipment Cost*

The equipment cost is expressed in terms of four cost factors: price, weight (mass), power consumption and computational load).

```
% Equipment cost
cost.price     = 20000;
cost.weight    = 0.3;
cost.power     = 1;
cost.cpu       = 0.1;
```

3.4.4 Configuration of Models

The configuration of the analytic redundancy relation models takes place in `configModels.m`. The following settings are possible:

► *Model Cost*

The cost of including an (ARR) model is expressed mainly in terms of computational load. Nevertheless, the other three factors can also be defined for it.

```
% Model cost
cost.cpu = 0.1;
```

3.5 Running the Analysis

Once the first configuration is finished, the analysis can be ran. The general approach is (skip steps if already executed):

- Open MATLAB 2016b
- Navigate to the folder “Gafe”
- Start-up GAFE by executing the function

```
> gafeStartup()
```

- Execute the following command to make the project just created the current project to work on:

```
> gafeProjectStartup()
```

- Run the structural analysis by executing the function

```
> runGafeStruct()
```

  Once the analysis is finished the results are displayed in the command window, are illustrated as plot (if desired) and are stored to files (log, xlsx and mat, see next section for details).

3.6 Analysis Outputs

The GAFE Structural Analysis provides the following type of outputs:

- Command line output with analysis results
- A diary file of the command line output (store under `bin\diary.log`)
- An excel file containing the analysis results (same as the one printed to command line if `showResultsTable = 1`, stored under `bin\resultsTable.xlsx`)
- A mat file containing the relevant data (process input parameters, analysis results) form the MATLAB workspace (stored under `bin\results.mat`)

After the analysis has finished, the results of the best solution found (and others, if demanded by user) are displayed in the command window and are illustrated graphically. The output in the command window contains the following information:

► Solution Overview

The required hardware set and the overall cost of the best solution:

```
Current Solution:
       Solution ID: 1
       Total Cost : 14.10
```

```
Required Hardware Set:
      mag: 3
      rmu: 2
      str: 2

Costs of Units:
      Unit Name        CPU [-]     Power [W]     Mass [Kg]   Price [EUR]
      mag1               0.100         1.000          0.30       20000.0
      mag2               0.100         1.000          0.30       20000.0
      mag3               0.100         1.000          0.30       20000.0
      rmu1               0.200        20.000          5.00      400000.0
      rmu2               0.200        20.000          5.00      400000.0
      str1               0.200         6.000          1.50      350000.0
      str2               0.200         6.000          1.50      350000.0
```

► Mode Overview

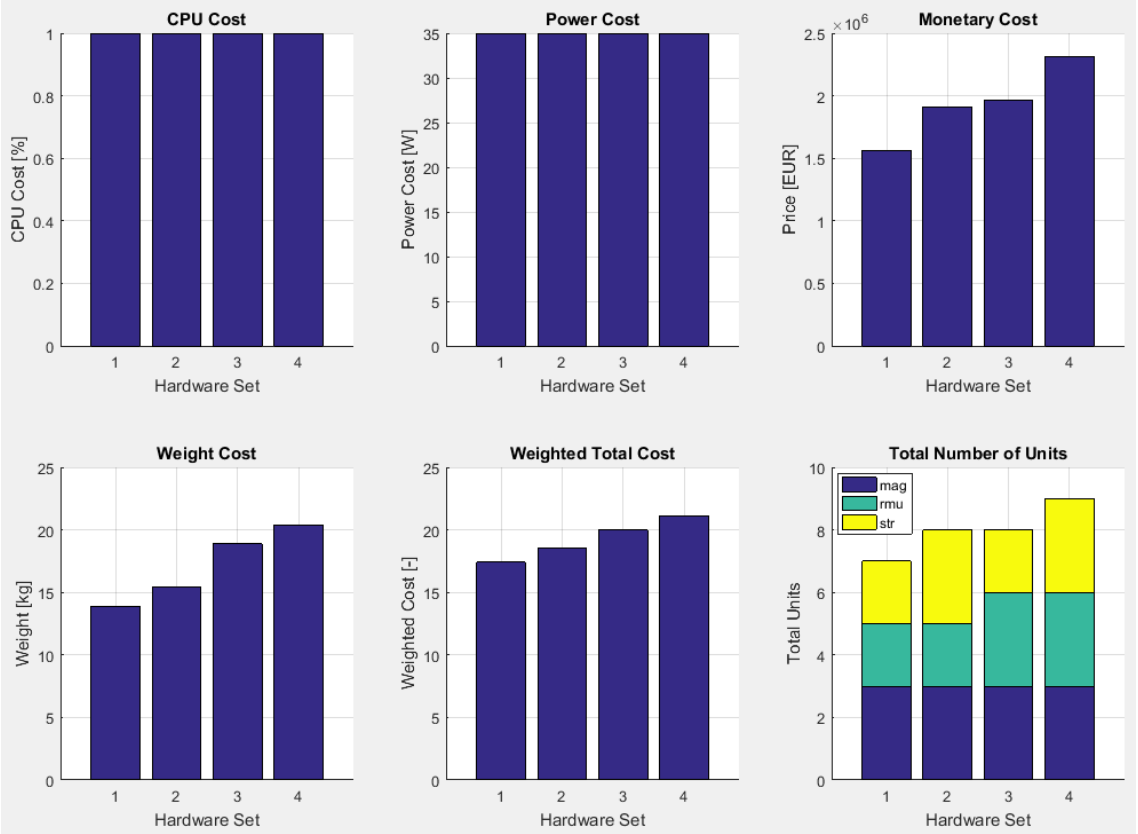
For each investigated AOCS mode the required active hardware, the required residuals to detect/isolate faults and the signatures for all relevant faults.

```
Best Active Hardware:
      mag: 3
      rmu: 1
      str: 0

Residuals:
      Residual 01:
            Relation(s)   : "rEomRot01AngAcc2AngRate_B",
"rEomRot01SumOfAngAcc_B",…
            Known State(s): "yRmu01AngRate_B"
      Residual 02:
            Relation(s)   : "rMag02Meas", "rMag03Meas"
            Known State(s): "yMag02MagField_B", "yMag03MagField_B"
      Residual 03:
            Relation(s)   : "rMag01Meas", "rMag03Meas"
            Known State(s): "yMag01MagField_B", "yMag03MagField_B"
      Residual 04:
            Relation(s)   : "rMag01Meas", "rMag02Meas"
            Known State(s): "yMag01MagField_B", "yMag02MagField_B"

Fault Signatures:
```

```
      Signature of "fMag01Meas":   --    --   (03)  (04)
      Signature of "fMag02Meas":   --   (02)   --   (04)
      Signature of "fMag03Meas":   --   (02)  (03)   --
      Signature of "fRmu01Meas":  (01)   --    --    --
```

► Cost Overview

If desired (by setting plotCostOverview = 1 in configAnalysis.m), a graphical overview of the costs and total number of units is given for all investigated solutions:



► Results Table

If desired (by setting showResultsTable = 1 in configAnalysis.m) a table with the status of all evaluated solutions is displayed:

Index	mag	rmu	stt	AocsModeA	AocsModeB	HWset	BestActiveHW_AocsModeA	BestActiveHW_AocsModeB	FD_FDI_AocsModeA	FD_FDI_AocsModeB	CPUcost	PowerCost	WeightCost	PriceCost	TotalCost	BestSolution
[6]	[0]	[1]	[1]	'-'	'OK'	'Active'	'-'	'-'	'nonValid'	'FDI'	[0.4000]	[26]	'-'	'-'	'-'	'-'
[7]	[0]	[1]	[2]	'-'	'OK'	'Active'	'-'	[6]	'nonValid'	'FDI'	[0.4000]	[26]	'-'	'-'	'-'	'-'
[8]	[0]	[1]	[3]	'-'	'OK'	'Active'	'-'	[6]	'nonValid'	'FDI'	[0.4000]	[26]	'-'	'-'	'-'	'-'
[10]	[0]	[2]	[1]	'-'	'OK'	'Active'	'-'	[6]	'nonValid'	'FDI'	[0.4000]	[26]	'-'	'-'	'-'	'-'
[11]	[0]	[2]	[2]	'-'	'OK'	'Active'	'-'	[6]	'nonValid'	'FDI'	[0.4000]	[26]	'-'	'-'	'-'	'-'
[12]	[0]	[2]	[3]	'-'	'OK'	'Active'	'-'	[6]	'nonValid'	'FDI'	[0.4000]	[26]	'-'	'-'	'-'	'-'
[14]	[0]	[3]	[1]	'-'	'OK'	'Active'	'-'	[6]	'nonValid'	'FDI'	[0.4000]	[26]	'-'	'-'	'-'	'-'
[15]	[0]	[3]	[2]	'-'	'OK'	'Active'	'-'	[6]	'nonValid'	'FDI'	[0.4000]	[26]	'-'	'-'	'-'	'-'
[16]	[0]	[3]	[3]	'-'	'OK'	'Active'	'-'	[6]	'nonValid'	'FDI'	[0.4000]	[26]	'-'	'-'	'-'	'-'
[21]	[1]	[1]	[0]	'OK'	'-'	'Active'	'-'	'-'	'nonValid'	'-'	'-'	'-'	'-'	'-'	'-'	'-'
[22]	[1]	[1]	[1]	'OK'	'OK'	'Active'	'-'	'-'	'nonValid'	'nonValid'	'-'	'-'	'-'	'-'	'-'	'-'
[23]	[1]	[1]	[2]	'OK'	'OK'	'Active'	'-'	'-'	'nonValid'	'nonValid'	'-'	'-'	'-'	'-'	'-'	'-'
[24]	[1]	[1]	[3]	'OK'	'OK'	'Active'	'-'	'-'	'nonValid'	'nonValid'	'-'	'-'	'-'	'-'	'-'	'-'
[25]	[1]	[2]	[0]	'OK'	'-'	'Active'	'-'	'-'	'nonValid'	'nonValid'	'-'	'-'	'-'	'-'	'-'	'-'
[26]	[1]	[2]	[1]	'OK'	'OK'	'Active'	'-'	'-'	'nonValid'	'nonValid'	'-'	'-'	'-'	'-'	'-'	'-'
[27]	[1]	[2]	[2]	'OK'	'OK'	'Active'	'-'	'-'	'nonValid'	'nonValid'	'-'	'-'	'-'	'-'	'-'	'-'
[28]	[1]	[2]	[3]	'OK'	'OK'	'Active'	'-'	'-'	'nonValid'	'nonValid'	'-'	'-'	'-'	'-'	'-'	'-'
[29]	[1]	[3]	[0]	'OK'	'-'	'Active'	'-'	'-'	'nonValid'	'nonValid'	'-'	'-'	'-'	'-'	'-'	'-'
[30]	[1]	[3]	[1]	'OK'	'OK'	'Active'	'-'	'-'	'nonValid'	'nonValid'	'-'	'-'	'-'	'-'	'-'	'-'
[31]	[1]	[3]	[2]	'OK'	'OK'	'Active'	'-'	'-'	'nonValid'	'nonValid'	'-'	'-'	'-'	'-'	'-'	'-'
[32]	[1]	[3]	[3]	'OK'	'OK'	'Active'	'-'	'-'	'nonValid'	'nonValid'	'-'	'-'	'-'	'-'	'-'	'-'
[37]	[2]	[1]	[0]	'OK'	'-'	'Active'	'-'	'-'	'nonValid'	'nonValid'	'-'	'-'	'-'	'-'	'-'	'-'
[38]	[2]	[1]	[1]	'OK'	'OK'	'Active'	'-'	'-'	'nonValid'	'nonValid'	'-'	'-'	'-'	'-'	'-'	'-'
[39]	[2]	[1]	[2]	'OK'	'OK'	'Active'	'-'	'-'	'nonValid'	'nonValid'	'-'	'-'	'-'	'-'	'-'	'-'
[40]	[2]	[1]	[3]	'OK'	'OK'	'Active'	'-'	'-'	'nonValid'	'nonValid'	'-'	'-'	'-'	'-'	'-'	'-'
[41]	[2]	[2]	[0]	'OK'	'-'	'Active'	'-'	'-'	'nonValid'	'nonValid'	'-'	'-'	'-'	'-'	'-'	'-'
[42]	[2]	[2]	[1]	'OK'	'OK'	'Active'	'-'	'-'	'nonValid'	'nonValid'	'-'	'-'	'-'	'-'	'-'	'-'
[43]	[2]	[2]	[2]	'OK'	'OK'	'Available'	'-'	[6]	'nonValid'	'FDI'	[0.4000]	[26]	'-'	'-'	'-'	'-'
[44]	[2]	[2]	[3]	'OK'	'OK'	'Available'	'-'	[6]	'nonValid'	'FDI'	[0.4000]	[26]	'-'	'-'	'-'	'-'
[45]	[2]	[3]	[0]	'OK'	'-'	'Active'	'-'	'-'	'nonValid'	'nonValid'	'-'	'-'	'-'	'-'	'-'	'-'
[46]	[2]	[3]	[1]	'OK'	'OK'	'Active'	'-'	'-'	'nonValid'	'nonValid'	'-'	'-'	'-'	'-'	'-'	'-'
[47]	[2]	[3]	[2]	'OK'	'OK'	'Available'	'-'	[6]	'nonValid'	'FDI'	[0.4000]	[26]	'-'	'-'	'-'	'-'
[48]	[2]	[3]	[3]	'OK'	'OK'	'Available'	'-'	[6]	'nonValid'	'FDI'	[0.4000]	[26]	'-'	'-'	'-'	'-'
[53]	[3]	[1]	[0]	'OK'	'-'	'Active'	'-'	'-'	'FDI'	'nonValid'	[0.8000]	[29]	'-'	'-'	'-'	'-'
[54]	[3]	[1]	[1]	'OK'	'OK'	'Active'	'-'	'-'	'FDI'	'FDI'	[0.7000]	[29]	'-'	'-'	'-'	'-'
[55]	[3]	[1]	[2]	'OK'	'OK'	'Active'	[54]	[54]	'FDI'	'FDI'	[0.7000]	[29]	'-'	'-'	'-'	'-'
[56]	[3]	[1]	[3]	'OK'	'OK'	'Active'	[54]	[54]	'FDI'	'FDI'	[0.7000]	[29]	'-'	'-'	'-'	'-'
[57]	[3]	[2]	[0]	'OK'	'-'	'Active'	[53]	'-'	'FDI'	'nonValid'	[0.8000]	[29]	'-'	'-'	'-'	'-'
[58]	[3]	[2]	[1]	'OK'	'OK'	'Active'	[54]	[54]	'FDI'	'FDI'	[0.7000]	[29]	'-'	'-'	'-'	'-'
[59]	[3]	[2]	[2]	'OK'	'OK'	'Available'	[53]	[6]	'FDI'	'FDI'	[0.8000]	[26]	[13.9000]	[1540000]	[14.1000]	[1]
[60]	[3]	[2]	[3]	'OK'	'OK'	'Available'	[53]	[6]	'FDI'	'FDI'	[0.8000]	[26]	[15.4000]	[1910000]	[15.2250]	[2]
[61]	[3]	[3]	[0]	'OK'	'-'	'Active'	[53]	'-'	'FDI'	'nonValid'	[0.8000]	[29]	'-'	'-'	'-'	'-'
[62]	[3]	[3]	[1]	'OK'	'OK'	'Active'	[54]	[54]	'FDI'	'FDI'	[0.7000]	[29]	'-'	'-'	'-'	'-'
[63]	[3]	[3]	[2]	'OK'	'OK'	'Available'	[53]	[6]	'FDI'	'FDI'	[0.8000]	[26]	[18.9000]	[1540000]	[16.7000]	[3]
[64]	[3]	[3]	[3]	'OK'	'OK'	'Available'	[53]	[6]	'FDI'	'FDI'	[0.8000]	[26]	[20.4000]	[2310000]	[17.8250]	[4]

3.7 Overview of Models

The GAFE Structural Analysis comes with a set of structural models for the most common AOCS sensors, AOCS actuators and analytical models for FDI purposes.

The input and outputs of these models are often “physical” quantities (e.g. the sensors output of a magnetorquer is a magnetic moment, instead of being the coil current). This is an abstraction but valid and useful in the frame of the structural analysis. Without knowing the relationship between current and magnetic moment, the measurement would not make that much sense for FDI purposes).

The complete list of models available in the GAFE Structural Analysis Tool is given in Section 5.1. In this section models of a magnetorquer and a LIDAR are briefly discussed in order to illustrate the concept and level of detail of structural models.

The different types of states and non-standard relations follow the graphical conventions illustrated in Figure 3-3.

In addition there are the following naming conventions:

- Known states are denoted by the prefix “y”
  - AOCS commands to units are denoted “u”, but they represent also known states.
- Unknown states are denoted by the prefix “x”
- Faults are denoted by the prefix “f”
- Relations are denoted by the prefix “r”

- Names of the equipment or analytical models and the index of the unit to which a certain item belongs are coded as follows:
  - “xMtq#” stands e.g. for an unknown state (x) of a Magnetorquer (Mtq). The unit index (#, e.g. 2 = Mtq2) is kept as variable which is filled automatically by the corresponding unit index during analysis.
- The rest of the name can be arbitrarily chosen, e.g. “uMtq#MagMomCmd_U”, which stands for the commanded magnetic moment in unit frame of a certain magnettorquer (where # is automatically replaced during analysis by the unit index).

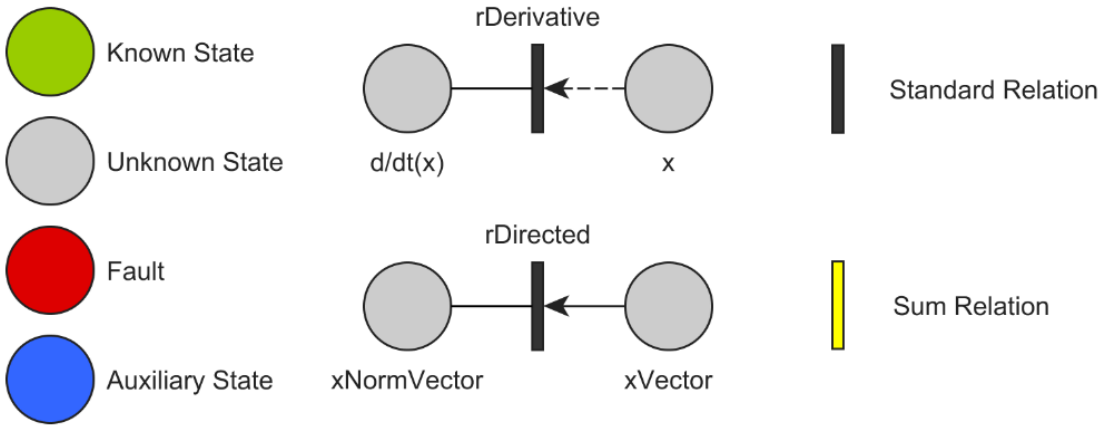


Figure 3-3: Overview of graphical conventions for states and relations.

► Magnetorquer

The structural model of a magnetorquer is illustrated in Figure 3-4. The known states are the commanded magnetic moment in unit frame (uMtq#MagMomCmd_U) and the measured magnetic moment in unit frame (yMtq#MagMomMeas, as it would be obtained via a model from the measured coil current).

The “Act” relation “rMtq#Act” represents the general relationship between a command of a certain quantity to an actuator and the realized quantity. That is why in this relation the fault is included. This states, that if this fault is present, one cannot conclude from the commanded quantity to the realized quantity.

The relation “rMtq#MagMon2Trq_U”, which has the current magnetic field of the Earth at the position of the spacecraft (e.g. obtained from an analytical magnetic field model fed by position measurements from a GNSR) as input, represents the physical behaviour: magnetic moment in magnetic field results in a torque.

The “rSumOfTrq_B” relation adds the obtained torque to the model of the rotational equations of motion. An additional measurement error “fMtq#Meas” could be added at the indicated place.

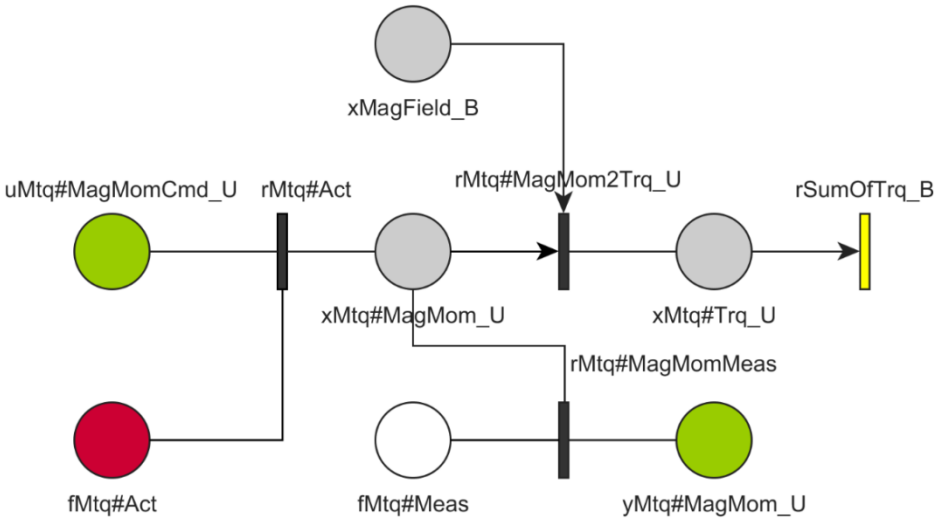


Figure 3-4: Structural Model of Magnetorquer.

► LIDAR (lidar)

The structural model of a LIDAR is illustrated in Figure 3-5. The known states are the measured relative position and relative attitude of the target with respect to the observers body frame (since attitude transformations between unit frames and the body frame are sufficiently know, there are often made no explicit distinctions, in order to keep the models campact).

Remark: The white circle indicates a possible second fault affecting the measured magnetic momentum only. Im both faults would be implemted simultaneously, the isolation of the two faults would require significant additional effort.

Each measurement (known state) has a measurement relation “r…meas”, which is connected to the “real”, i.e. unknow quantity. Instead of connecting the measurement fault of the LIDAR directly to both measurement relations, it is connected via an auxiliary relation (rLidar#AuxRel) to an auxiliary state (fLidar#AuxState), which then is connected to the two measurement realtions. The reason for this approach is that the used fault diagnosis toolbox does not allow a fault to enter directly in multiple relations. If such a modelling is required (mainly for sensors where the measuremet fault corrupts several measured quantities), the concept of auxiliary relation plus auxiliary state has proven useful.

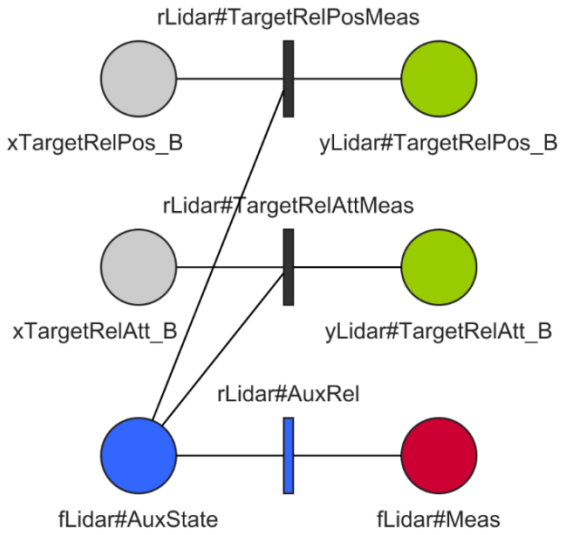


Figure 3-5: Structural Model of LIDAR.

3.8 Tool Extension

In order to adapt the tool to specific project needs which are not covered by the models contained in the GAFE library, the tool can be extended in the following ways:

- Add new equipment model (sensor or actuator)
- Add new analytic redundancy relation ARR model

The method off adding new analytic models is exactly the same as for a sensors or an actuator. Therefore the following approach is only described generically.

A good starting point for adding a new model (of sensor, actuator or analytic model) is to look at the models already existing. They are located under `src\models\`. The model for a star tracker looks e.g. like this:

```
classdef CSensorStr < CBasicItem
   methods
      function obj = CSensorStr(cost)

        % Create equipment as instance of CBasicItem
        obj@CBasicItem('str', 'Star Tracker', 'sensor', cost);

        % Define known states
        obj.knownStates   = {'yStr#Att_JB'};

        % Define unknown states
        obj.unknownStates = {'xAtt_JB'};

        % Define faults
        obj.faults        = {'fStr#Meas'};
```

```
        % Define relations
        obj.relations     = {'rStr#Meas', …
                             {'yStr#Att_JB', 'xAtt_JB', 'fStr#Meas'}};

      end
   end
end
```

It is derived from `CBasicItem`, which brings along properties for the short name (here `'str'`), the long name (here `'Star Tracker'`), the type (here `'sensor'`) and the cost structure with the fields `'cpu'`, `'weight'`, `'power'`, `'price'`.

The rest of the model file describes the known states, the unknown states, the faults, and - how all of them are connected - the relations.

► *Templates:* There exist templates for each type of model:

- `CSensorXxx.m`: for sensors
- `CActuatorXxx.m`: for actuators
- `CModelXxx.m`: for analytic models

By using the templates and having a look at the available models of the same type, a new user defined model can be created relatively easy.

► Dynamic/SumOf Relations: Special attention has to be paid if unknown states should affect unknown states of analytic models. This is at the moment the case for actuators, because all forces, linear accelerations, torques and angular accelerations generated by actuators are used in the dynamic models for translation (`'rEomTrans'`) and rotation (`'rEomRot'`). That means that these relations are dynamic, meaning that they look different if there are e.g. 3 RWs or 4 RWs included in the project (or in the current analysis step)

Therefore such “dynamic” relations have to be added also to new actuator models. At the moment there exist the following dynamic relations (also simply called “sumOf”-relations”):

- `'rEomTrans#SumOfAcc_J'`
- `'rEomTrans#SumOfFrc_B'`
- `'rEomRot#SumOfAngAcc_B'`
- `'rEomRot#SumOfTrq_B'`

Example: the model of the reaction control system (RCS) contains besides it normal “rRcs#Act” relation also the external dynamic relations `SumOfFrc_B` and `SumOfTrq_B`:

```
% Define relations
obj.relations = {'rRcs#Act', {'uRcs#FrcTrqCmd_B', 'xRcs#FrcTrq_B',…};
                 'rEomTrans#SumOfFrc_B', {{'xRcs#FrcTrq_B', 'IN'}};
                 'rEomRot#SumOfTrq_B',   {{'xRcs#FrcTrq_B', 'IN'}}};
```

► Integral/Differential and Directed Relations: Another important concept are directed relations, of which there exist three types:

- Integral/differential relations (`'D/I'`)
- Directed relations (`'IN'`)

Integral/differential relations describe the relation between a state and its derivative, e.g. position and velocity. The corresponding relation of such a relationship contains always exactly two states and look like:

```
% Define relations
obj.relations = {'rXxx#diRel', {{'vel', 'D'}, {'pos', 'I'}};
```

The GAFE Structural Analysis is configured such that it can conclude in the direction of the derivative, i.e. if the position is known, the velocity is also known if there is a corresponding D/I relation. The other way is by default not possible, since this would require the initial condition to be known in addition.

Directed relations describe a one-way-street relation between a state and another. If e.g. a 3D vector is known, it is easy to compute its norm. To compute the vector from its norm, on the other hand, is impossible. The corresponding relation look as follows:

```
% Define relations
obj.relations = {'rXxx#dirRel, {{'vec', 'IN'}, 'pos'};
```

Directed inputs can also occur in relations with more than 2 state, e.g.:

```
% Define relations
obj.relations = {'rXxx#abc, {{'a', 'IN'}, 'b', {'c', 'IN'}};
```

► Faults: any relation can contain faults. They have to be defined as fault and can then be included in relations like any other state:

```
% Define faults
obj.faults = {'fStr#Meas'};
```

4 GAFE – Simulator

4.1 Workflow

This section briefly describes the typical steps to be executed when working with the GAFE simulator.

- Run "`gafeStartup.m`" to start up the GAFE library.
- Create a new project (see section 4.4.2) or start up an existing one by executing "`gafeProjectStartup.m`" in the respective project folder.
- Create and develop own "Equipment" (see section 4.7) and "AOCS Algo" (see section 4.8) models if required in the addition to the default library models (see sections 4.6.2 and 4.6.3)
- Define the default parameterization for the simulator modules by adapting the values in the respective parameter files in the `\param` folders (see section 4.3)
- (Optionally) define tests in a separate folder (see sections 4.4.3 and 4.4.4). Tests can be used to define test case specific parameters which differ from the default parameterization of the scenario.
- Run a simulation of the default scenario after executing the function "`initGafeSim`" (without arguments) which automatically opens the GAFE Simulink model
- And/Or run a test either
  - automatically using the function `runTest` (see section 4.4.5)
  - manually by executing the desired `paramTest` files preceded by "`initGafeSim(TestPar)`"
- Post process the simulation data by visualization or specific procedures (see 0)

4.2 Parameter Inheritance

The GAFE parameter inheritance concept is as follows:

- The GAFE framework provides a baseline parameterization for all its modules. This parameterization is called "library parameters".
- The modifications performed by the user of the GAFE framework to adapt it to the project under investigation leads to the "default parameters".
  - Note: this part might involve intermediate inheritance steps, like e.g. from equipment to units, or AOCS modes to submodes.
  - Realization: Overwriting of "library parameters" by "default parameters" is implicitly performed by the precedence of files in the "`GafeProjects`" folders over file in the "`Gafe`" folder (see 2.2).
- The definition of tests is separated into 3 levels

- Test topic: used to sum up test cases belonging together, e.g. all test cases for a certain AOCS mode, mission phase or spacecraft configuration
  - Each test topic has a dedicated "`paramTest.m`" file, which overwrites entries of "default parameters".
- Test case: a specific test for a certain thing, e.g. the complete failure of a reaction wheel. A test case can consist of multiple test runs.
  - Each test case has a dedicated "`paramTest.m`" file, which overwrites entries of the result from the step before.
- Test run: each test run represents a dedicated simulation. Test runs are mainly used for parameter variations within a test case. These variation can either be deterministic (one run for each value of paramX = [1, 2, 3]) or probabilistic (paramX = 0.77*randn()). The last option is used for monte-carlo analysis, e.g. to vary the injection time of faults.

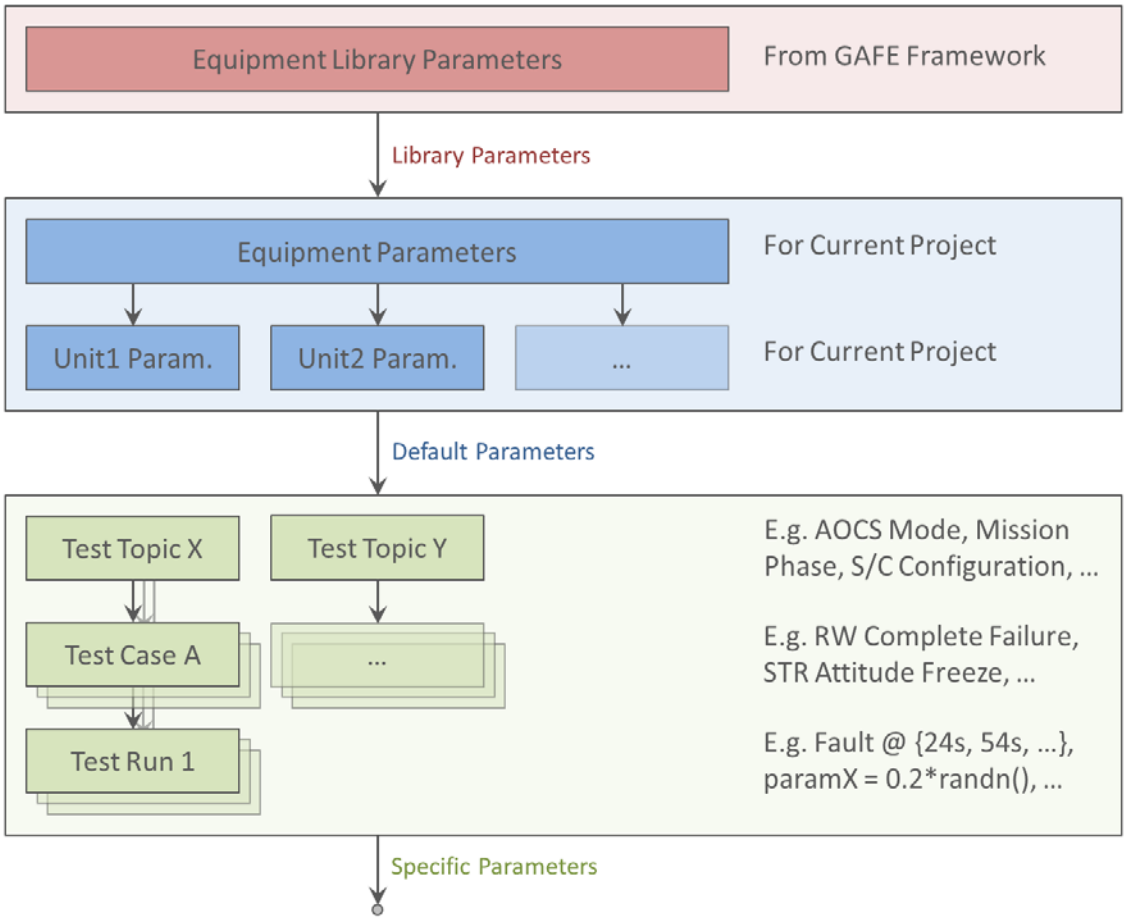


Figure 4-1: Parameter inheritance concept.

4.3 Parameterization & Initialization

4.3.1 Description

- Parameterisation data is separated from initialisation, in different files
  - Name of the individual parameter file is "`param<Name>.m`"
  - Name of the individual initialisation file is "`init<Name>.m`"
  - <Name> matches the name of the related model (i.e. AocsAlgo, System, Magnetometer, etc.).
- Parameters or initialisation that cannot be assigned to model part (i.e. sample time and end time of simulation or the test case name) are collected in
  - "`paramGeneral.m`" and
  - "`initGeneral.m`"
- Further initialization and parameterization files exist for
  - Each Module
  - Each Equipment
  - + depending on needs
- Parameterisation data in the files above are treated as default values. For modifying the default values for a particular test case an additional specific parameter file called "`paramTest.m`" is used, which is not in the MATLAB path. There are separated "`paramTest.m`" files for each test case, potentially also additional "`paramTest.m`" files per test topic (see section 4.4).
- The parameterisation of AOCS Algorithms contains also the configuration of AOCS Algorithms, read therefore section 4.3.2
- Initialisation files are called hierarchically
- Each Initialisation file is broken down into the following steps
  - Evaluate parameterisation file "`param<Name>.m`"
  - Overload obtained values with "`paramTest.m`"
  - The obtained user parameter structure is called "`UserPar`"
  - The actual initialisation file transforms the user parameter structure "`UserPar`" into the simulation parameter structure "`SimPar`"
- Finally the structure "`SimPar`" is the one and only variable which is used for GAFE simulation

4.3.2 Configuration of AOCS Algorithms

Each piece of the reusable set is called "AOCS Algorithm Component" or shortly "Component" if the context is clear.

The overall AOCS algorithms will be set up by scheduling already defined Components. This is done by the user in the file "paramAocsAlgo.m".

This configuration will be used to generate the "aocsAlgo.m" main function which calls all referenced AOCS Algorithms Components. "aocsAlgo.m" will be generated in the “bin” folder when initialising the simulator.

First make sure you have all these components. A library of predefined Component is available for the user, see chapter 5.2, however the user may develop new ones according to the given rules and verify them. See chapter 4.8.

Detailed rules how to configure are listed in “paramAocsAlgo.m”. This includes also a how to assign AOCS Algorithms Components states to the SGM (more about SGM you can read in Section 4.6.5.1).

However see here as example the AocsAlgo configuration of the GafeDemo,

```
% I/O scheduling of the used components
aocsAlgoCfg = { ...
   % "loopNum"   "funName"       "inputList"

   NaN,    'AocsClass',          'Sensor Processing'; ...
   'Mag',  'magMeasProc(:)',     {'UnitsHk.Mag(:)', 'UnitsStatusExpected.Mag(:)'}; ...

   NaN,    'AocsClass',          'Determination'; ...
   1,       'magFieldEst',        {'magMeasProc','UnitsStatusExpected.Mag'}; ...

   NaN,    'AocsClass',          'Controller'; ...
   1,      'asmMagRateDampCtrl', 'magFieldEst'; ...

   NaN,    'AocsClass',          'Actuator Commanding'; ...
   1,      'dynCmdDistribution', {'asmMagRateDampCtrl', 'aocsMode'}; ...
   'Mtq',  'mtqCmd(:)',          {'UnitsStatusExpected.Mtq(:)', 'dynCmdDistribution',
                                  'magFieldEst'}; ...
   'Mtq',  'Assign',             {'UnitsCmds.Mtq(:).mtqMagMomCmd',
                                  'mtqCmd(:).magMomCmd', 'mtqCmd(:).isValid'}; ...
};

% Assign components to modes

aocsAlgoModeUse.magMeasProc =       {'*'};
aocsAlgoModeUse.magFieldEst=        {'ASM*'};

aocsAlgoModeUse.asmMagRateDampCtrl= {'ASM_RD'};
aocsAlgoModeUse.dynCmdDistribution= {'ASM*'};
aocsAlgoModeUse.mtqCmd=             {'ASM*'};
```

and an example (see also chapter 5.2.2.3) of the project specific parametrisation. Note, that user needs to parametrise only those items which deviate from the default parametrisation as given for all Components listed in chapter 5.2

```
%%
%=========================================================================
% Determination

%---------------------------------------
% Magnetic Field

UserAocsAlgo.MagFieldEst.numUnits = 3;     % This parameter might be fed automatically
                                           % from Equipment Config ...
UserAocsAlgo.MagFieldEst.method = 1;       % 1: selection, 3: middle value of each
                                           % component
UserAocsAlgo.MagFieldEst.selSeq = [2,1,3];
pdiffTimeConst = 6; % [s]
sampleTime = aocsSampleTime;
```

```
UserAocsAlgo.MagFieldEst.pdiffParams = computePdiffParams(pdiffTimeConst, sampleTime);
UserAocsAlgo.MagFieldEst.rateLowLim = 0.1*pi/180;  % [rad/s]
```

Both snippets are extracted from corresponding “paramAocsAlgo.m”.

4.3.3 Rules

- For Modules and Components: Parameterisation matches input and output w.r.t. use of busses and structure. This means if input and output are busses then parameterisation is a structure. For sure there are cases where this rule cannot be applied
- Structure “arrays” are allowed for
  - Equipment Units
  - AOCS Algorithms processing of Sensors and Actuators
  - “System” Module
  - FDIR
- For structure array
  - Use “non-virtual-bus” and define it using “createBus.m” to generate a file “loadBusDef_[busName].m”
- Structures and buses start with a capital letter, all other variables (vector, matrices, etc.) start lower case; IDs in definition structures are completely upper case (e.g. DefAocsMode.NOM);
- Use 0/1 instead of true/false in Simulink signals/buses

4.4 Project and Test Case Organization

4.4.1 Concept

- Test cases can be organized in an arbitrary folder hierarchy (though starting from a fixed location ‘<projectFolder>\Simulator\data\test’). This hierarchy can e.g. be used to organize test cases by test topics
- All leaf-folders represent test cases, all folders above test topics (potentially several levels)
- Each level in the hierarchy contains a single "paramTest.m" file, which contains the specific parameters for this level
- The priority of specific parameters is bottom-up:
  - The definition of a parameter (e.g. Q = 4) in the "paramTest.m" of a test case has precedence over its definition in (one of) the "paramTest.m" of the test topic(s) above (Q = 3, Q = 2, …).

4.4.2 Creating a GAFE Project

To create a project, call the function createProject.

Call it with one argument to generate the project at place

```
createProject(projectName)
```

or with two arguments specifying also the folder path

```
createProject(projectName,projectFolder)
```

A folder with the project name with all necessary subfolders and files will be created.

This folder structure reflects the folder structure of the simulator library, but with only parameterization files present. If a runtime or initialization file is intended to be substantially changed, it can be copied to or recreated at the corresponding location in the project. This file then shadows the equivalent library function.

All existing parameterization files are added to the “param”-folder. It is for your awareness and it is recommended to browse through it, and if you know that you won’t use it, just delete it. However if you need it later, you can just take it from the GAFE library ... as it is just a copy.

4.4.3 Create Tests

To create a test, a utility function `createTest` can be used. The function can be used either for preparing a single test or a complete “hierarchy” of tests in a single step. In the first case, syntax is as follows:

```
createTest('<testName>')
```

where `testName` is as sting for the test name to be created. This call generates a subfolder in the current working directory which contains a template “`paramTest.m`“ to define the test parameters (see also next section). Note that no check on the current directory is performed, but test shall be located in below in `\<projectDir\Simulator\data\param\test\`).

For the second case, the syntax is as follows:

```
createTest(testFolderStruct)
```

with

- testFolderStruct

  A Matlab structure representing the tree structure for the test folders. Field names and subfield names correspond to the names of test topics and test cases.

`createTest` creates all the test topic & test case folders with the exact same folder names and structure as in `testFolderStruct`. Inside each test folder, a “`paramTest.m`“ file is created which serves as template to define test topic and test case specific parameters (see also next section). Note that this option overwrites all existing “`paramTest.m`“ files that might already be present in the provided folder structure.

4.4.4 Defining Test Topic & Test Case Specific Parameters

To define topic and/or test case specific parameters, simply add the desired parameter definition (w.r.t. to the library values) in the "`paramTest.m`" file on the desired level. Test parameter values can either be specified explicitly or generated randomly as further described below.

4.4.4.1 Deterministic Test Parameters

The assignment to the parameter structure needs to match the following form:

```
TestPar. <...> = <value>;
```

where <...> is the respective part of the User<...> parameter structure from the GAFE "library" parameter file (in …`\data\param\`).

Examples:

Overwrite the fundamental simulation sampling time parameter "`UserGeneral.simSampleTime`" specified in `paramGeneral.m`:

```
TestPar.General.simSampleTime = 1;
```

Overwrite the default value of the nominal measurement noise standard deviation (band-limited white noise, BLWN) for all accelerometers specified in paramAccelerometer ("`UserAcc.Default.SigMa.satNonGravAccMeas_U.Blwn.std`"):

```
     TestPar.Acc.Default.SigMa.satNonGravAccMeas_U.Blwn.std = ...
[1;1;1;]*1e-4;
```

4.4.4.2 Randomized Test Parameters

Randomized test parameter (i.e. parameters which shall be varied between different simulation runs of the test) are generated using the utility function `generateTestParam` in the following form:

```
TestPar.<...> =
generateTestParam(nominalValue,numTestRuns,varMethod,varParam,seed);
```

where <...> is the respective part of the User<...> parameter structure from the GAFE "library" parameter file (in …`\data\param\`). The inputs to `generateTestParam` are:

- nominalValue

  The nominal value of the input parameter to which the random variation is additively applied.

- nTestRuns

  The number of test runs (which is also the number of random samples to be generated).

Note: If nTestRuns is not unique over all levels (i.e. all paramTest.m files) or parameters used in a test, always the smallest identified nTestRuns is effectively used.

- varMethod

  A string for the variation method to be applied (‘`Gaussian`‘ or ‘`Uniform`)

- varParam (optional)

  An 1x2 double array which represents the distribution parameter values of the chosen `varMethod`: [lower bound, upper bound] in case of a uniform distribution or [mean, standard deviation] for the Gaussian.

  This is argument is optional. If omitted, standard uniform [0,1] and standard Gaussian [0,1] distributions are used.

- ‘seed’

  The random number generator seed assigned for this random parameter.

Example:

Randomly vary the magnetic field measurement bias of all magnetometers uniformly in a range of ±1000nT over 67 test runs using a seed 1234:

```
TestPar.Mag.Default.SigMa.magFieldMeas_U.Bias.value =
generateTestParam([2; 2; 5]*1e-6, ,67,'Uniform',[-1e-6,1e-6],1234);
```

4.4.5 Running Tests

To execute tests, the function "`runTest`" can be used in several ways:

- Run the test case in the current working directory (taking into account all test parameter from the root test directory down to the current directory):

```
runTest()
```

- Run the test case in the current working directory (for only a subset of the number of test runs):

```
runTest(testRunIndex)
```

- Run all test cases of current project (without “test case path” argument):

```
runTest('all')
```

- Run test cases belonging to a specific test topic:

```
runTest(folderStruct)
```

- Run a single test case or a range of test cases:

```
runTest(folderStruct, testRunIndex)
```

- Run selected test cases in subfolders (with option ‘all’ or ‘explicit’):

```
runTest(folderStruct, subTestOption)
```

or

```
runTest(folderStruct, testRunIndex, subTestOption)
```

with:

- folderStruct

  A Matlab structure which represents the part of the test folder hierarchy levels (starting below `<projectPath>data\test\`) which shall be included in the test run. The field and subfield names in `folderStruct` must correspond to a consecutive path in `\test\` to the desired sublevel or sublevels.

- testRunIndex

  A positive integer number (or a range of numbers), indicating which single test (or range of tests) out of `nTestRuns` (see section 4.4.4.2) should be conducted following the `folderStruct` test folder route.

- subTestOption

  A string describing the further test options:

  - ‘all’

    All tests present in the given `folderStruct` will be conducted, including all subfolders below the lowest level folder defined in `folderStruct`

  - ‘explicit’ (default)

    Only the test folders explicitly defined in `folderStruct` will be conducted.

Combining `testRunIndex` with `subTestOption` ‘explicit’, allows conducting only the n-th test (specified by `testRunIndex`) in the test folders explicitly specified in `folderStruct`; together with `subTestOption` ‘all’, the n-th test in all subfolders of `folderStruct` is conducted.

Example (see Figure 4-2):

```
testStructure.TestTopicAbc.TestCaseAAA = [];
runTest(testStructure,…)
```

This first overwrites the project’s default parameters by the specific parameters defined in the "`paramTest.m`" of the test topic (`'TestTopicAbc'`), and then by the specific parameters defined in the "`paramTest.m`" of the actual test case (`'TestCaseAAA'`). Other test topics and test cases are ignored as they are not part of the structure. Similarly, folders “on the route” which do not contain a `paramTest.m` are ignored.

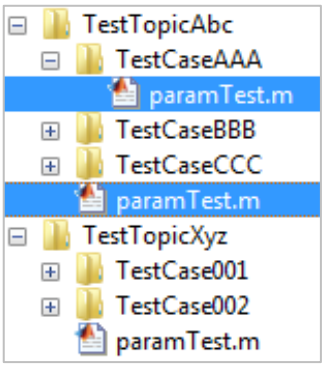


Figure 4-2: Folder structure of test cases in a project.

The result of each test run is automatically saved in `<projectPath>bin\test\` in a `.mat` file. The folder structure mirrors the folder structure of the test definition in `<projectPath>data\test\`. The name of the `.mat` file corresponds to name of the test folder in which the test parameters are located and is suffixed by the index of the test run (i.e. `TestCaseAAA_run001.mat` is saved in `\bin\test\TestTopicAbc\TestCaseAAA` for the example above). The stored. The data stored in file comprises the test, user and simulation parameter structures (`TestPar`, `SimPar`, `UserPar`) and well as the actual Simulink signals logged to the workspace as `Simulink.SimulationData.Dataset` object named "`simLog`". The latter can be used for visualization or further post-processing.

4.4.6 Post-Processing

After running a test from the test folder the command loadTest(testRunIndex) can be used to load the test data to the base workspace.

4.4.6.1 Visualizing Simulation Data

After test data has been loaded, the command plotGafeSim() can be used to open the tool plotFields, which lists all logged simulation data in a hierarchical view (which mirrors the hierarchy of the GAFE Simulator itself). From this tool all simulation data can be accessed and analysed.

4.4.6.2 Specific Post-Processing

After test data has been loaded, all simulation data (and the used parameterization) are available in the workspace. The simulation data is contained in the structure "simLogStruct", which can be used to write specific post-processing routines.

4.5 Concepts Used in GAFE Simulator

4.5.1 Definitions

Adopting the concept of "#define" directives from c-language, definitions are used in GAFE to enable the use of mnemonics and constants. The basic idea is to define items in one central place and then use them consistently in different places (e.g. in the parameter file, the initialization file and the run time code of a component or also across different components).

Typical applications are e.g. to define mnemonics for AOCS modes or to define physical or mathematical constants.

4.5.1.1 Definition File

The actual definition shall take place in one definition file per set of items belonging together. The name of such a file shall be `def<SetName>.m`. and it shall be stored under …`\data\def\`. Two exemplary definition files are given below:

```
function def = defAocsModeIds()

    % Mnemonic of AOCS mode IDs
    Def.STB = 1;
    Def.ASM = 2;
    Def.NOM = 3;
    Def.OCM = 4;

end
```

```
function def = defPhysicalConstants()

    % Physical constants
    Def.speedOfLight      = 299792458.0;
    Def.BoltzmannConstant = 1.3806485279e-23;
    Def.meaningOfLife     = 42.0;

end
```

4.5.1.2 Use of Definition File

In order to use definitions made in definition files they must be made available to functions they shall be used in. In parameter files, initialization files, etc. this can be simply done by calling the corresponding define function:

```
DefAocsModeIds = defAocsModeIds();
```

In code of run time functions of the GAFE Simulator persistent variables shall be used to store definitions:

```
function output = myFunction(velocity, aocsMode)

    persistent isFirstCall DefAocsModeIds DefPhysicalConstants

    if isempty(isFirstCall)

        % Assign definitions to hierachical structure
        DefAocsModeIds       = defAocsModeIds();
        DefPhysicalConstants = defPhysicalConstants();

        isFirstCall = false;

    end;
```

```
    if (velocity > DefPhysicalConstants.speedOfLight && ...
        aocsMode == DefAocsModeIds.NOM)
        % warp
    end;

end
```

4.5.2 Persistent Variables

Persistent variables inside run time functions are initialized with the following mechanism:

```
function output = myFunction(input, varAaa0, varBbb0)

    persistent isFirstCall varAaa varBbb

    if isempty(isFirstCallState)

        % Assign definitions to hierarchical structure
        varAaaState = varAaa0;
        varBbbState = varBbb0;

        isFirstCallState = false;

    end;

    %...

end
```

4.5.3 Simulations

- Logging of simulation data is performed via “logging antennas” which are placed in the model during model development
- For each antenna the logging decimation is 1.
- If a single bus comes out of the model this bus is named identical as the model itself, which leads to a consistency when a logging antenna is put on the bus
- Simulink.SimulationData.Dataset is used for logging. The name of the logging parameter is: “`simLogModelDataSet`”
- After simulation a conversion is performed into a classical structure, which is better suited for postprocessing, by call of the dedicated function `simulinkDatasetToStruct`. The name of the logging structure is called “`simLog`”

4.6 GAFE Simulator Modules

In this section module and component specific items are collected together with a general description of the modules.

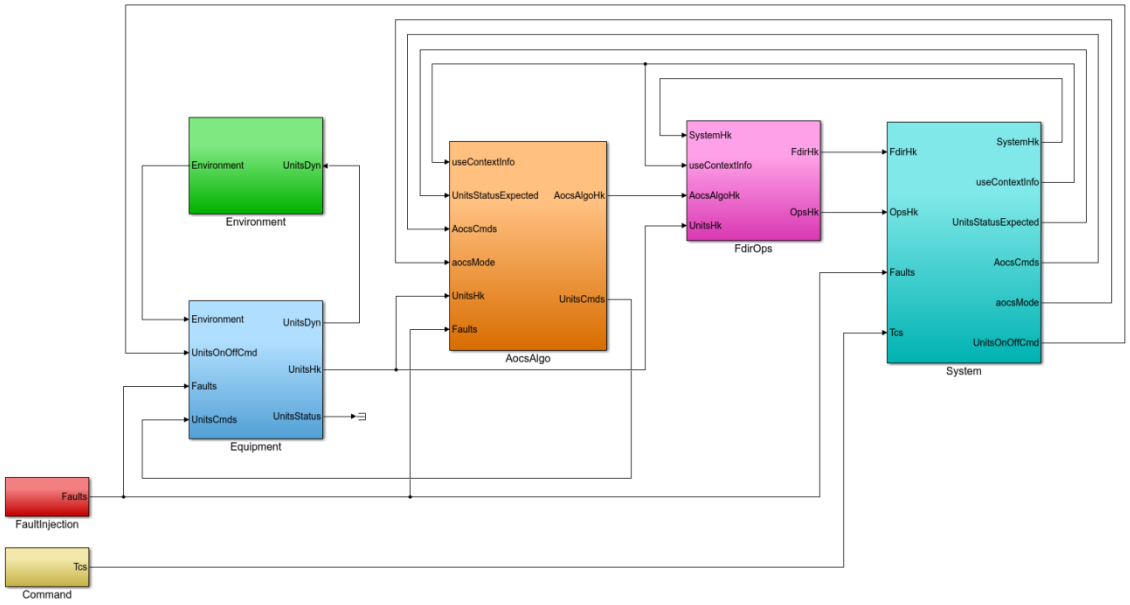


Figure 4-3: Top-level architecture of the GAFE Simulator.

4.6.1 Environment

The Environment module consists of several components which are required to compute the spacecraft motion and to provide all “ideal” environmental data as input to the equipment models. The parameterization of the module is entirely defined in the file “`paramEnvironment.m`”. (in …`\data\param\`) and its structure reflects the present components which are briefly described in the following.

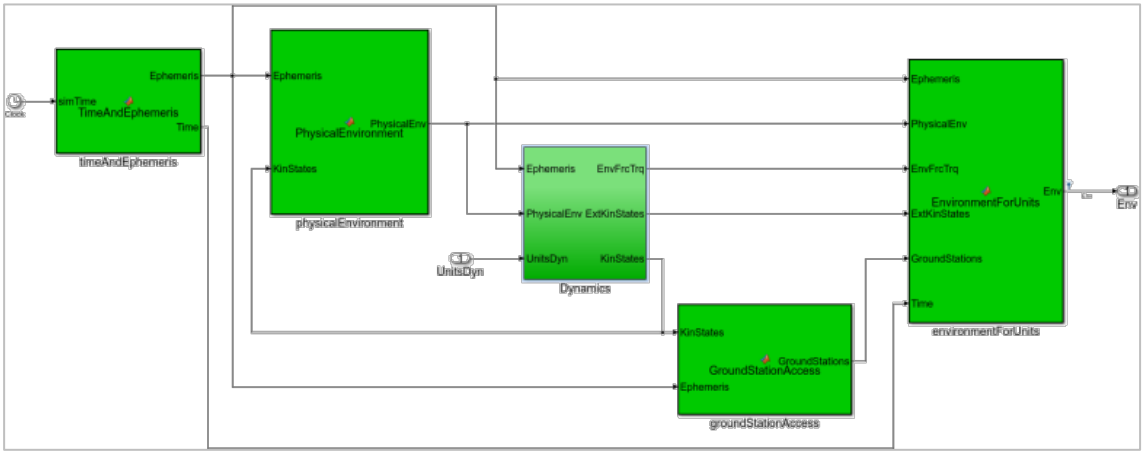


Figure 4-4: Components of Environment module.

- TimeAndEphemeris

  The component parameterization allows setting the initial date for a simulation in different time formats and time scales. Further, flags can be set to consider or neglect Earth precession and nutation models.

  The output bus provides the current time in various formats, sun and moon position as well as the attitude between different Earth-fixed reference frames (E,Q,M,J).

- PhysicalEnvironment

  This component allows parameterizing the environmental models for

  - Gravity: degree and order of a spherical harmonics model
  - Atmosphere: inclination factor related to a Harris-Priester density model
  - Magnetic field: degree, order and reference year of a spherical harmonics model
  - Eclipse: cylindrical or conical shadow model

  Note: functions for other models related to gravity (spherical gravity, J2) and magnetic field (dipole) exist as well to be used for “model-based approaches” in the AocsAlgo module.

  The output bus provides an eclipse flag, the atmospheric density, the inertial magnetic field vector as well as gravity accelerations from Earth, sun and moon.

- Dynamics

  The component parameterization allows setting the initial translational and rotational state in different formats (e.g. Cartesian or Keplerian coordinates; attitude angles, DCM or quaternions) as well as mass, inertia and satellite geometry and frame properties. Further, flags can be set to consider or neglect the effect of certain environmental disturbances on the S/C motion.

  The output bus provides the “extended” S/C states comprised of position, attitude, (linear and angular) velocity & acceleration and angular momentum vectors

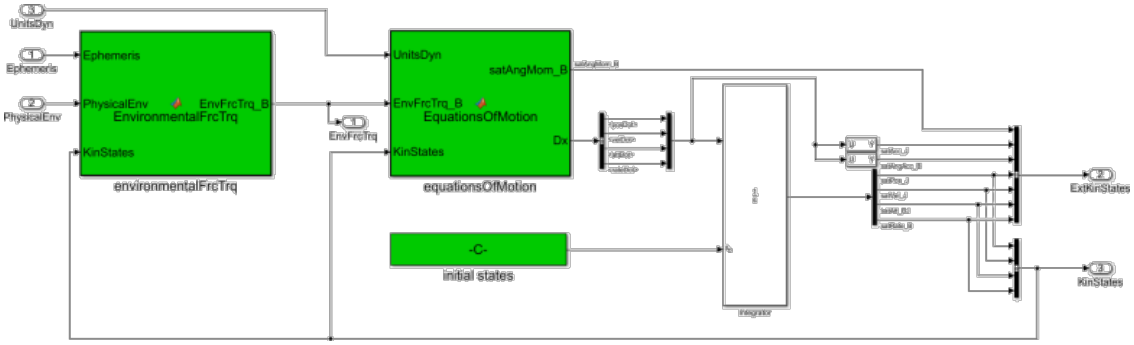


Figure 4-5: Dynamics subcomponents.

- GroundStationAccess

  The component parameterization allows defining the Earth-fixed positions and minimum elevations for a customizable number of ground stations.

The output bus provides range, range rate, azimuth, elevation and visibility information of the spacecraft w.r.t. all defined stations.

- EnvironmentForUnits

  This component basically serves as “bus collector” for all previous environment module components only.

4.6.2 Equipment

The Equipment module consists of a single component “GenericEquipmentModel” (GEM). This component covers all generic parts (selection of applicable signal manipulations for fault/nominal cases, operational states and transitions) and calls the equipment specific core algorithms (the “specific equipment models” ,SEM).

Note, when selecting and changing the number of equipments (or units of a certain equipment), no manual change of the Simulink model is required.

The top-level parameterization in paramEquipment.m (in …`\data\param\`) only selects the number of units for all defined equipment types which shall be used in a simulation. The parameterization of a specific equipment model uses a dedicated file param[EquipmentName].m (in …`\data\param\SEM\`). Please refer to section 4.7 for a detailed description on how to create, modify and parameterize equipment models.

The Equipment module bus outputs the dynamics contribution of equipments (actuators) to the S/C dynamics, housekeeping data (available to the AOCS algorithms and the FdirOps module) and auxiliary status outputs which can be used for post-processing and analysis of a simulation run.

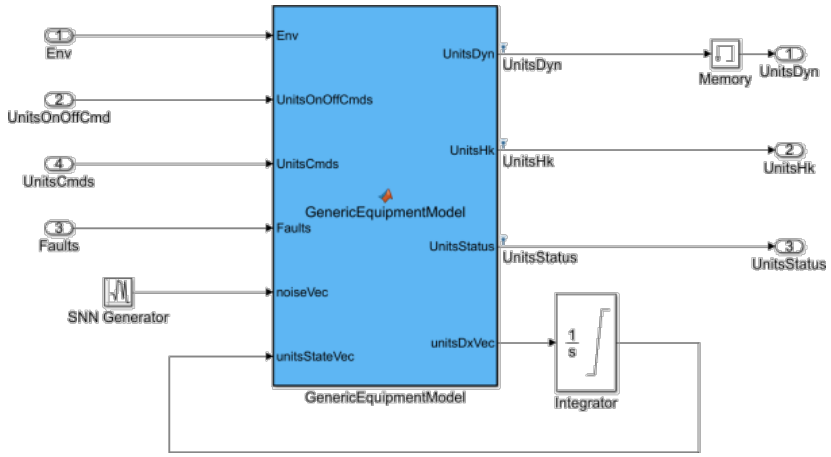


Figure 4-6: Components of Equipment module.

In order to ease the parameterizion of geometrical properties for quipment models (like e.g. alignment and position), a generic set for spacecraft and unit coordintate systems has been defined (see Figure 4-7).

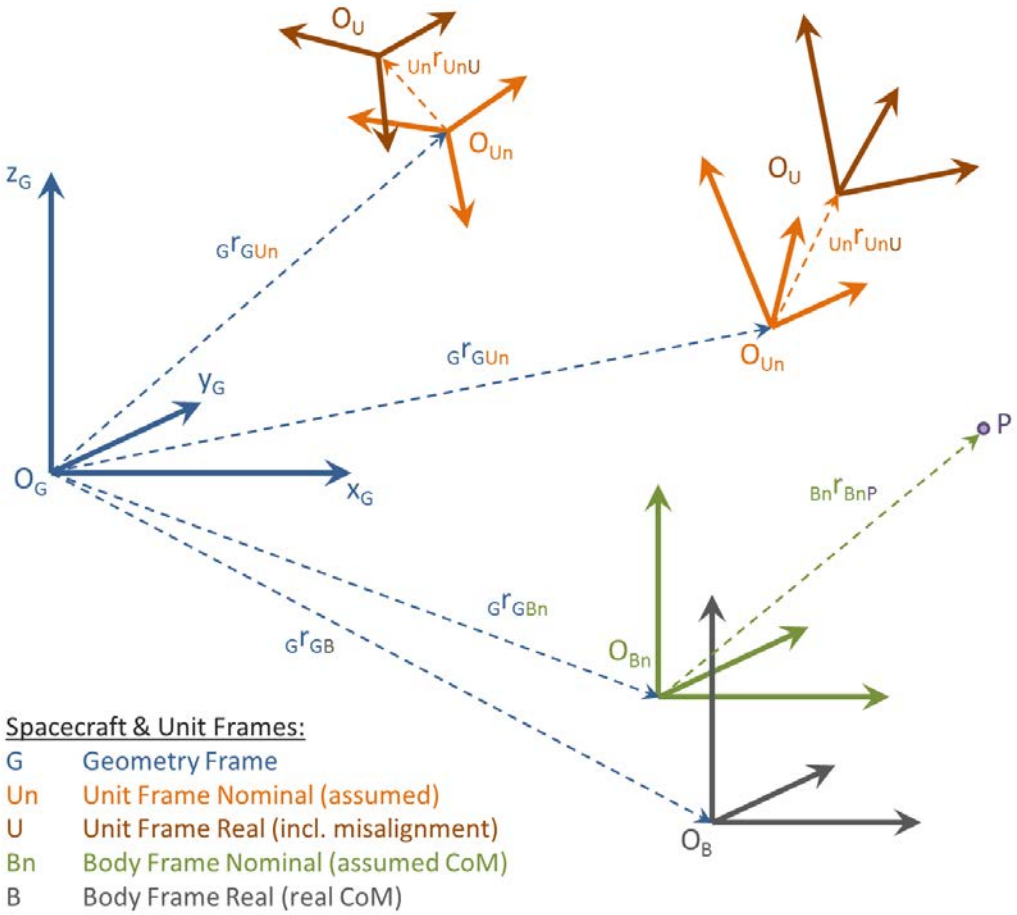


Figure 4-7: Definition of spacecraft and unit coordinate systems.

The GAFE library includes the following specific equipment models by default:

4.6.2.1 Sensors

Accelerometer

This equipment represents a 3-axis sensor which outputs the non-gravitational acceleration acting on the spacecraft. When displaced from the spacecraft center of mass, couplings from gravity-gradient (simplified model), angular accelerations and angular velocity contributions are considered. Bias, different noise models, scale errors and output saturation is modeled for the nominal operation.

Camera

This equipment represents a visual navigation sensor which measures the relative direction and relative distance to a target in the vicinity of the investigated spacecraft. The following effects

are taken into account to determine if a valid output can be provided: blinding of sensor by the sun (sun in field of view), target illumination, target in field of view and target in distance.

Earth Sensor

This equipment represents a direction sensor which measures the unit vector from the unit to the Earth center. Blinding by the sun, maximum spacecraft rates and distance to the Earth are taken into account to determine if a valid output can be provided. Further, blinding by the sun leads to a degraded performance. Measurement noise is applied during nominal operation.

GNSS Receiver

This equipment represents a global navigation satellite system receiver which provides Earth-fixed position and velocity vectors of the spacecraft together with the GPS time as main outputs (additional housekeeping outputs are the navigation mode and GDOP and TDOP values). It takes into the account the number and orientation of visible GNSS satellites (based on the receiver field of view and simplified orbit propagation and occultation models). Further, different performances can be modeled independently to represent the acquisition and nominal operation phase (i.e. navigation mode) of the sensor. Noise can applied to all main measurement outputs.

LIDAR

This equipment represents a LIDAR which measures the relative position and relative attitude of a target in the vicinity of the investigated spacecraft. The following effects are taken into account to determine if a valid output can be provided: is the target in field of view, is the target in a certain distance, is the relative velocity exceeded and is the relative rate exceeded.

Magnetometer

This equipment represents a 3-axis sensor which outputs the measured magnetic field of the Earth at its location. Non-orthogonality of the measurement axes as well as measurement bias, noise, scale errors and saturation are applied to the output during nominal operation.

Rate Measurement Unit

This equipment represents a 3-axis sensor which outputs the measured angular rate of the spacecraft at its location. Non-orthogonality of the measurement axes as well as measurement bias, noise, scale errors, quantization errors and saturation are applied to the output during nominal operation.

Star Tracker

This equipment provides the inertial attitude of the unit expressed as quaternion vector together with a measurement quality index and the measurement (center of integration) time stamp. Blinding by the sun or the earth together with user-defined rate limits are used to determine if a valid output can be provided. Further, blinding by the moon and the sensor acquisition time determine if parameters for full or reduced performance parameters for the noise are applied to compute the output signals.

Sun Sensor

This equipment represents a direction sensor which outputs the unit vector from the unit to the sun center. Blinding by the Earth, and maximum spacecraft rates are used to determine if full performance or reduced performance are applied for the noise models and if a valid output can be provided at all.

4.6.2.2 Actuators

Magnetic Torquer

This equipment represent a single torquer unit which translates a given magnetic moment command into a torque applied on the spacecraft body.

Reaction Control System (Thrusters)

This equipment represent an assembly of thrusters which takes a commanded force and/or torque command vector (i.e. no individual thrusters are modeled or can be commanded) as input. The commanded input is perturbed by noise on the direction and magnitude during nominal operation

Reaction Wheel

This equipment represent a single reaction wheel which takes a torque command as input (no wheel speed can be directly commanded). The model accounts for both torque and wheel speed limits and includes coulomb and viscous friction models. Biases and scale factor errors are applied to the realized torque and measured wheel speed outputs during nominal operation.

4.6.2.3 Other Equipment

Antenna Pointing Mechanism

This equipment represents a simplified antenna pointing mechanism model which takes a 2D angular rate command as input and provides a 2D angular measurement together with an antenna pointing direction vector as output. The measurement outputs are corrupted with noise, bias and scale errors as well as quantization during nominal operation. The maximum rate can also be limited and quantized. The main simplification in the model is that is does not affect the spacecraft dynamics at all (i.e. no change in angular momentum, torque or inertia).

Feared Event Actuator (FEA)

This equipment is an artificial device to introduce disturbance forces, torques and/or angular momenta (single or any combination of) acting on the spacecraft body. During nominal operation, this “equipment” has no effect unless a fault event is defined which introduces such disturbance. It is useful for direct stimulation of FDIR mechanisms designed to handle feared events like e.g. excessive spacecraft rates or attitude anomalies.

In order to use the FEA the number of units of this actuator has to be set to one. Afterwards corresponding fault(s) can be injected into it. Available faults are ANGMOM, FRC, TRQ. The magnitude and type of disturbance (bias, noise, random walk, etc.) for each of those faults can be set by parameters.

OBC Clock

This equipment is a simple provider of the OBC time as Modified Julian Date (GPS time scale). In nominal operation, bias, noise and scale errors can be applied to the output.

Solar Array Drive Mechanism

This equipment represents a simplified solar array drive mechanism model which takes an angular rate command as input and provides a measured angle as output. The measurement is corrupted with noise, bias and scale errors as well as quantization during nominal operation. The maximum rate can also be limited and quantized. The main simplification in the model is that is does not affect the spacecraft dynamics at all (i.e. no change in angular momentum, torque or inertia).

4.6.3 AocsAlgo

The AocsAlgo module hosts all AOCS algorithms, including algorithms for AOCS related FDIR, such as analytical model and residual generators.

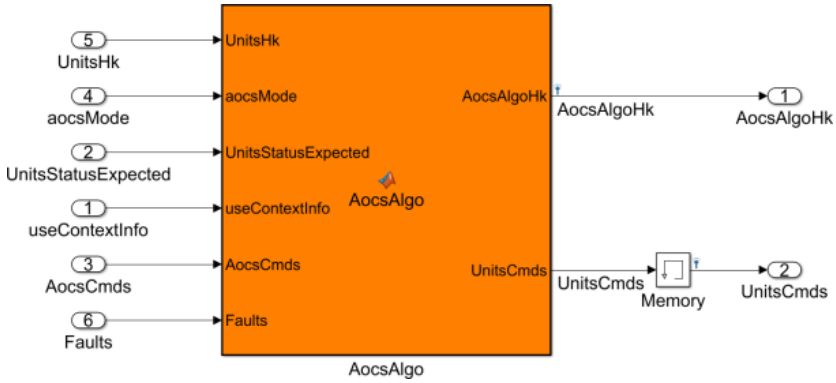


Figure 4-8: Components of AocsAlgo module.

4.6.4 FdirOps

The FdirOps Module provides the essential FDIR mechanisms for parameter monitoring (PUS Service 12) and functional monitoring (PUS Service 143, Airbus proprietary). Additionally, it includes the core functions of PUS Service 5 (Event Reporting Service) and 19 (Event Action Service). The definition of the parameter and functional monitors is done directly via MATLAB structures, which allows a very flexible parameterisation, if desired under revision control.

The core element of the FdirOps Module, instantiated as FdirServices and as OpsServices, is the so-called FDIR Module. This module was developed early in the GAFE study and is already used in multiple AIRBUS projects. Therefore it has a separate technical documentation & user guide, which can be found under [RD-5].

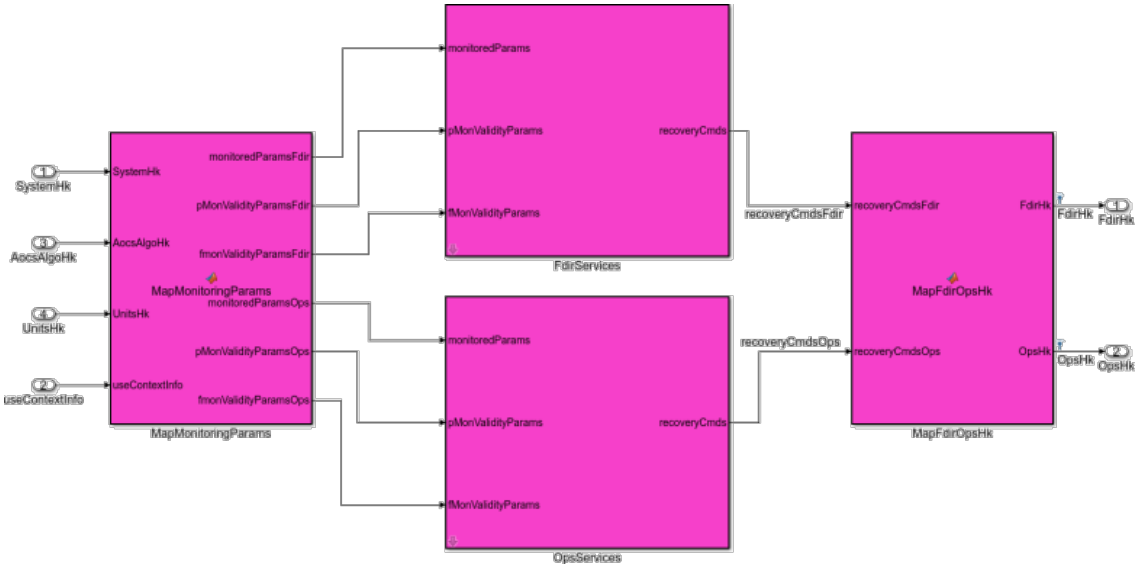


Figure 4-9: Components of FdirOps module.

4.6.4.1 Main Overservables

The most important variables (“observables”) to monitor or to use as validity parameters for monitoring are listed hereafter sorted by origin. Items with fields “`(EquipmentName)(UnitIdx)`” are available or each unit of each equipment.

Example:

```
SystemHk.EquipmentManagerHk.UnitsStatusExpected.Mag(1).isOp
SystemHk.EquipmentManagerHk.UnitsStatusExpected.Lidar(2).isOp
```

► System

```
% AOCS Mode Manager
SystemHk.AocsModeManagerHk.aocsModeCurrent
SystemHk.AocsModeManagerHk.aocsModeDesired
SystemHk.AocsModeManagerHk.aocsMainModeCurrent
SystemHk.AocsModeManagerHk.aocsMainModeDesired
SystemHk.AocsModeManagerHk.modeDuration

% Equipment Manager
SystemHk.EquipmentManagerHk.UnitsStatusExpected.(EquipmentName)(UnitIdx).isPwr
SystemHk.EquipmentManagerHk.UnitsStatusExpected.(EquipmentName)(UnitIdx).isCom
SystemHk.EquipmentManagerHk.UnitsStatusExpected.(EquipmentName)(UnitIdx).isOp
SystemHk.EquipmentManagerHk.UnitsStatusExpected.(EquipmentName)(UnitIdx).isUsbl
SystemHk.EquipmentManagerHk.EquipmentStatus.availabilityCurrent
SystemHk.EquipmentManagerHk.EquipmentStatus.availabilityDesired
SystemHk.EquipmentManagerHk.EquipmentStatus.readinessCurrent
SystemHk.EquipmentManagerHk.EquipmentStatus.readinessDesired
SystemHk.EquipmentManagerHk.UnitsOnTime.(EquipmentName)

% System Configuration Manager
SystemHk.SystemConfigManagerHk.systemState
SystemHk.SystemConfigManagerHk.processorModule
SystemHk.SystemConfigManagerHk.avionicChain
SystemHk.SystemConfigManagerHk.initialAocsMode
SystemHk.SystemConfigManagerHk.unitsPowerCycling
SystemHk.SystemConfigManagerHk.useContextInfo
SystemHk.SystemConfigManagerHk.enableFdir
SystemHk.SystemConfigManagerHk.systemConfigId
```

```
SystemHk.SystemConfigManagerHk.systemOnTime
SystemHk.SystemConfigManagerHk.systemOnTime
```

► Equipment

```
% Unit Status for power and communication (PCDU and
% Communication Surveilance are not modelled in GAFE;
% therefore information is directly obtained from
% equipment models)
EquipmentHk.UnitsHk.(EquipmentName)(UnitIdx).isPwr
EquipmentHk.UnitsHk.(EquipmentName)(UnitIdx).isCom

% Validity information from "intelligent equipment"
EquipmentHk.UnitsHk.(EquipmentName)(UnitIdx).isMeasValid

% HK information from (usually intelligent) equipment
EquipmentHk.UnitsHk.(EquipmentName)(UnitIdx).posQualIndex
EquipmentHk.UnitsHk.(EquipmentName)(UnitIdx).timeQualIndex
EquipmentHk.UnitsHk.(EquipmentName)(UnitIdx).navigationMode
```

► AocsAlgo (overview is not complete; for a full list see Section 5.2)

```
% Sensor Processing: General validity
AocsAlgoHk.AocsAlgoStatus.(EquipmentName)MeasProc(UnitIdx).isValid

% Sensor Processing: Unit specific validities
AocsAlgoHk.AocsAlgoStatus.StrMeasProc(UnitIdx).isValid
AocsAlgoHk.AocsAlgoStatus.StrMeasProc(UnitIdx).isValidAtt
AocsAlgoHk.AocsAlgoStatus.StrMeasProc(UnitIdx).isValidRate
AocsAlgoHk.AocsAlgoStatus.StrMeasProc(UnitIdx).isProcNewData
AocsAlgoHk.AocsAlgoStatus.StrMeasProc(UnitIdx).isMeasDeclaredValid
AocsAlgoHk.AocsAlgoStatus.StrMeasProc(UnitIdx).isFiniteData
AocsAlgoHk.AocsAlgoStatus.StrMeasProc(UnitIdx).isLivingData
AocsAlgoHk.AocsAlgoStatus.StrMeasProc(UnitIdx).isInLimTimeDif
AocsAlgoHk.AocsAlgoStatus.StrMeasProc(UnitIdx).isInLimSatAttQuatNorm

% Estimation/Determination: General validity
AocsAlgoHk.AocsAlgoStatus.TimeEst.isValid
AocsAlgoHk.AocsAlgoStatus.EphemerisDet.isValid
AocsAlgoHk.AocsAlgoStatus.SsSunDirEst.isValid
AocsAlgoHk.AocsAlgoStatus.SsSunDirEst.isValidSunDir
AocsAlgoHk.AocsAlgoStatus.SsSunDirEst.isValidSunRate
AocsAlgoHk.AocsAlgoStatus.SsSunDirEst.isUsblAny
AocsAlgoHk.AocsAlgoStatus.SsSunDirEst.isSunDetected
AocsAlgoHk.AocsAlgoStatus.SsSunDirEst.isUsed
AocsAlgoHk.AocsAlgoStatus.RmuSatRateEst.isValid
AocsAlgoHk.AocsAlgoStatus.RmuSatRateEst.isUsblAny
AocsAlgoHk.AocsAlgoStatus.RmuSatRateEst.isUsed
AocsAlgoHk.AocsAlgoStatus.StrSatAttEst.isValid
AocsAlgoHk.AocsAlgoStatus.StrSatAttEst.isValidAtt
AocsAlgoHk.AocsAlgoStatus.StrSatAttEst.isValidRate
AocsAlgoHk.AocsAlgoStatus.StrSatAttEst.isCrossCheckOk
AocsAlgoHk.AocsAlgoStatus.StrSatAttEst.isUsblAny
AocsAlgoHk.AocsAlgoStatus.StrSatAttEst.isAttFused
AocsAlgoHk.AocsAlgoStatus.StrSatAttEst.isRateFused
AocsAlgoHk.AocsAlgoStatus.StrSatAttEst.isRateHold
AocsAlgoHk.AocsAlgoStatus.StrSatAttEst.isUsed
AocsAlgoHk.AocsAlgoStatus.Oop.isValid
AocsAlgoHk.AocsAlgoStatus.Oop.isUsblAny
AocsAlgoHk.AocsAlgoStatus.Oop.isProp
AocsAlgoHk.AocsAlgoStatus.Oop.isUsed
AocsAlgoHk.AocsAlgoStatus.MagFieldEst.isValid
AocsAlgoHk.AocsAlgoStatus.MagFieldEst.isValidMagField
AocsAlgoHk.AocsAlgoStatus.MagFieldEst.isValidMagRate
AocsAlgoHk.AocsAlgoStatus.MagFieldEst.isCrossCheckOk
AocsAlgoHk.AocsAlgoStatus.MagFieldEst.isUsblAny
AocsAlgoHk.AocsAlgoStatus.MagFieldEst.isMagRateLow
AocsAlgoHk.AocsAlgoStatus.MagFieldEst.isUsed
…
```

► Ground Station

```
Environment.GroundStations.visibility
```

4.6.4.2 Assistance Functions

The exist a few assistance function for FDIR and OPS related monitoring, which allow to setup a certain mechanism in a very handy way. The following mechanisms are available:

► FDIR

- addFdirPowerNok: Adds for a single AOCS unit a set of monitors (pMon and fMon) and recovery actions to the FDIR, which monitor the status of the unit's power (current) survailance function of an AOCS unit. This function is normally part of the PCDU, in the GAFE simulator it is modelled as part of the generic equipment model).

- addFdirComNok: Adds for a single AOCS unit a set of monitors (pMon and fMon) and recovery actions to the FDIR, which monitor the status of the unit's communication (TM/TC) survailance function of an AOCS unit. This function is normally part of the data handling, in the GAFE simulator it is modelled as part of the generic equipment model).

- addFdirMeasProcNok: Adds for a single AOCS unit a set of monitors (pMon and fMon) and recovery actions to the FDIR, which monitor if the validity ("isValid") of the unit's measurement processing inside the AOCS algorithms does becomes not okay ("NOK"). Use for short term detection of measurement anomalies.

- addFdirMeasProcNotOk: Adds for a single AOCS unit a set of monitors (pMon and fMon) and recovery actions to the FDIR, which monitor if the validity ("isValid") of the unit's measurement processing inside the AOCS algorithms does stay not okay (not "OK") for longer. During nominal unavailability of a unit (e.g. a sun sensor during eclipse) the "isValid" information of this unit has the status "NotEvaluated", since no measurements are available. The set of monitors added by addFdirMeasProcNotOk checks if the duration of such – normally expected situation - does not persist to long.

- addFdirCrossCheckIntraEquipmentNok: Adds for all AOCS units of the same equipment type a set of monitors (pMon and fMon) and recovery actions to the FDIR, which monitor the cross-checks between all the active units of this equipment. E.g. the cross-check between all the sun directions measured by the sun sensors, or the magnetic field magnitude and direction of all magnetometers.

- addFdirCrossCheckInterEquipmentNok: Adds for all AOCS units of two equipment types a set of monitors (pMon and fMon) and recovery actions to the FDIR, which monitor the cross-checks between all the active units of these two equipment types. E.g. the cross-checks between the spacecraft rates of all STRs and RMUs.

- addFdirEquipmentAvailabilityNone: Adds a set of monitors (pMon and fMon) and recovery actions to the FDIR, which monitor the availability of the equipment set for the current AOCS mode (explained in more detail in the discussion of the equipment manager above).

► OPS

- addOpsAocsModeAutoTransition: Adds a set of monitors (1x pMon and 1x fMon) and an action to the OPS, which perform a AOCS mode transition once the given transitions conditions are fulfilled for a given time span.

Please refer to the headers of these files for the exact purpose and syntax.

4.6.5 System

The System module represents all functions of the on-board computer which have direct impact on the AOCS, but do not belong to AOCS itself. As shown in Figure 4-10 it consists of the following components:

- System Configuration Manager
- Equipment Manager
- AOCS Mode Manager

The covered functionalities and parameterization options of each component are discussed in the following subsections.

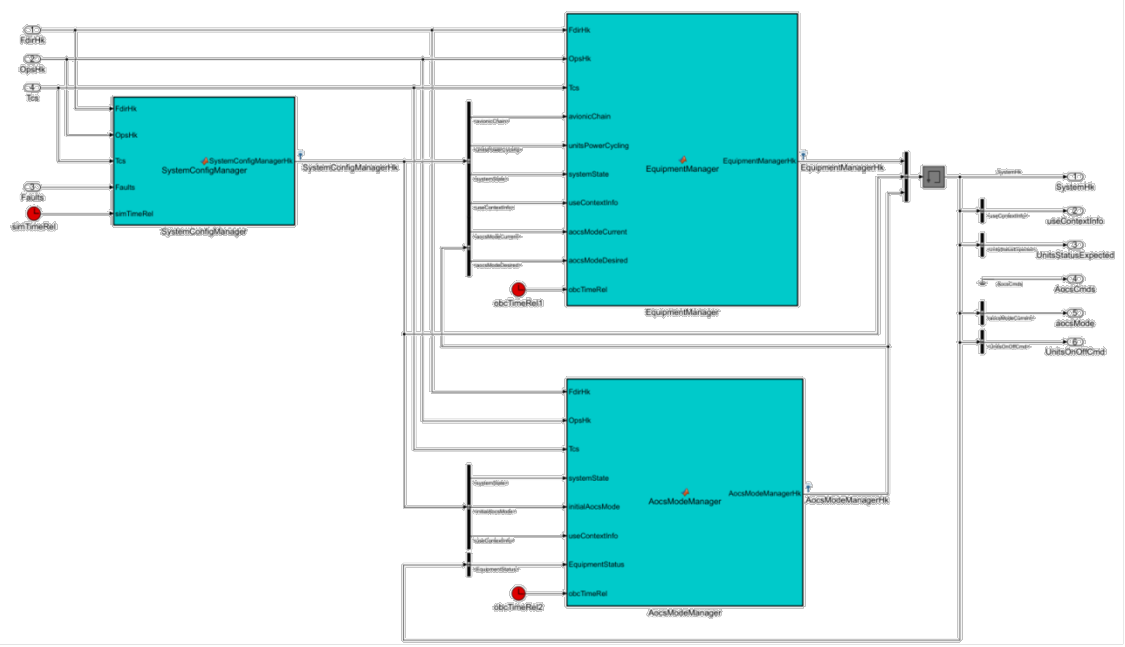


Figure 4-10: Components of the System module.

4.6.5.1 System Configuration Manager

The System Configuration Manager (`SystemConfigManager`, see Figure 4-11) is in charge of providing basic system level information (the so-called “system configuration”) to all receiving components.

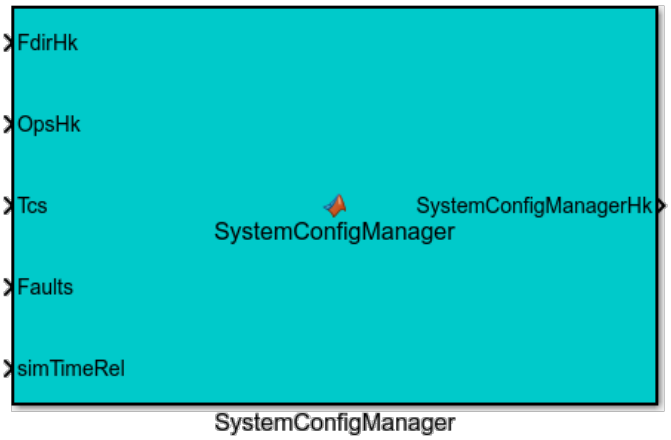


Figure 4-11: System Configuration Manager.

A system configuration consists of the following items

- systemConfigId: the ID of the system configuration

  - Possible values are positive integer numbers (consecutive for multiple configurations
- nextSystemConfigId: the ID of the system configuration to use after a system reconfiguration, e.g. a reboot or a “nextSystemConfig” action from FDIR or OPS.
  - Possible values are the IDs of all defined system configurations (self-referencing and cyclic-referencing are allowed)
- processorModule: the processor module to use
  - Possible values are e.g. 'A', 'B', …
- avionicChain: the avionic chain to use.
  - Possible values are e.g. 'A', 'B', …
- initialAocsMode: the AOCS mode to use initially or after reboot,
  - Possible values are e.g. 'ASM', ‘NOM’, …
- unitsPowerCycling: indication whether to power cycle AOCS equipment after reboot:
  - Possible values are: 0 (no), 1 (yes)
- useContextInfo: information whether to use context information stored in non-volatile memory (e.g. for propagators or ephemeris models) when entering a new system configuration, e.g. after a reboot:
  - Possible values are: ‘NONE’, ‘SGM’, ‘RAM’
- enableFdir: Flag which indicates if the FDIR shall be enabled or disabled
  - Possible values are: 0 (no), 1 (yes)

The system configuration manager stores this information in a tabular way, allowing to easily define multiple system configurations (see Table 4-1).

Table 4-1: System configuration table.

System Config ID	Next System Config ID	Processor Module	Avionic Chain	Initial Aocs Mode	Units Power Cycling	Use Context Info	Enable FDIR
1	2	A	A	ASM	No	Yes	Yes
2	3	A	B	ASM	No	Yes	Yes
3	4	B	A	SAME	Yes	Yes	Yes
4	5	B	B	ASM	Yes	No	Yes
5	6	A	A	ASM	Yes	No	Yes
6	1	B	A	ASM	Yes	No	No

The System Configuration Manager is also the component that simulates the initial start-up of the system or the restart after an OBC reboot. During the start-up the systemState is “INI”(tialization) and after that “OP”(erational). The boot duration and elapsed boot time can be set via parameters. During the (re)start the system configuration information is not available to the other components, which are anyhow not executed, since the system state is not yet operational.

4.6.5.1.1 Parameterization

The System Configuration Manager can be configured with the set of parameters explained hereafter. They can be set in the `paramSystem.m` file in the corresponding section. Please notice that the options for many items are predefined in dedicated “def”-lists, of which most can be modified by the user (e.g. `defAocsModeIds()`, `defAvionicChainIds()`, `defProcessorModuleIds()`):

► System Configuration Table: entries as explained above.

```
% Configuration 1 (mandatory)
iSysConf = 1;
SystemConfigTable(iSysConf).nextSystemConfigId = 2;
SystemConfigTable(iSysConf).processorModule    = DefProcessorModuleIds.A;
SystemConfigTable(iSysConf).avionicChain       = DefAvionicChainIds.A;
SystemConfigTable(iSysConf).initialAocsMode    = DefAocsModeIds.FF_SK;
SystemConfigTable(iSysConf).unitsPowerCycling  = 1;
SystemConfigTable(iSysConf).useContextInfo     = DefContextInfoIds.SGM;
SystemConfigTable(iSysConf).enableFdir         = 1;

% Configuration 2
iSysConf = iSysConf + 1;
SystemConfigTable(iSysConf).nextSystemConfigId = 3;
SystemConfigTable(iSysConf).processorModule    = DefProcessorModuleIds.B;
SystemConfigTable(iSysConf).avionicChain       = DefAvionicChainIds.B;
…
```

► System Configuration Manager Sample Time: normally the sampleTime of the System Module.

```
% Sample time of System Configuration Manager
sampleTime = systemSampleTime;
```

► Boot duration

```
% Duration of OBC (re)boot [s]
bootDuration = 60;
```

4.6.5.1.2 Initial Conditions

In addition to the parameterization the System Configuration Manager offers the initial conditions discussed hereafter. Like the parameterization, they can also be set in the `paramSystem.m` file in the corresponding section:

► Initial System Configuration

```
% ID of initial system configuration (ID must match a config defined
above)
systemConfigId0 = 1;
```

► Elapsed Boot Time

```
% Elapsed time since last reboot [s]
% (If larger than boot duration, OBC starts in operational state,
otherwise in initialization)
elapsedBootTime0 = 30;
```

4.6.5.2 Equipment Manager

The Equipment Manager (`EquipmentManager`, see Figure 4-12) is in charge of AOCS unit activation and deactivation. This includes nominal operation (like e.g. activation of required units at system start-up, activation of new units before a mode transition and deactivation of not required units after the transition) but also includes all unit reconfigurations resulting from FDIR actions. In addition to the unit switch-on/off commands the Equipment Manager reports information about the general availability of the equipment set for the current and desired mode, and if any activation of units in currently in progress.

The following items are contained in the components output `EquipmentManagerHk`:

- `UnitsStatusExpected`
  - The expected status of all AOCS unit in terms of the four "`isXxx`" status information: `isPwr, isCom, isOp, isUsbl`
  - In principle the status is purely based on expectation (defined by assumptions on the durations each unit requires to switch its operational mode from off, via initialization to operational) and its main purpose is to generate the validity information required by FDIR to activate the corresponding monitoring functions. E.g. once the expected status `isOp` of MAG3 becomes true, the monitoring on the validity information of the MAG3 measurement processing within the AOCS algorithms can start.
  - `UnitsStatusExpected.(EquipmentName)(unitIdx).isXxx`
- `UnitsOnOffCmd`
  - Current commands to the AOCS units
  - `UnitsOnOffCmd.(EquipmentName)(unitIdx).isXxx`
  - Possible value from `defUnitCmdIds()`
- `UnitsOnOffCmdPending`
  - Pending commands to the AOCS units (for next cycle)
  - `UnitsOnOffCmdPending.(EquipmentName)(unitIdx).isXxx`
  - Possible value from defUnitCmdIds()
- `UnitsHealth`

  - The current health of all units (the health of a unit is decreased by FDIR by a unit failed actions)
  - `UnitsHealth.(EquipmentName)(unitIdx)`
  - Possible values: 0 = unitFailed, x = unit ok and x restarts/power cyclings allowed
- `UnitsOnTime`
  - The on-times of all units since their last start/restart in seconds
  - `UnitsOnTime.(EquipmentName)(unitIdx)`
- `EquipmentStatus.availabilityCurrent`

  `EquipmentStatus.availabilityDesired`
  - General availability of the AOCS equipment set for the current/desired AOCS mode, i.e. if there is any valid unit configuration (based on unit configuration table and current health status of units) for all equipment types
  - Possible values (from `DefAvailabilityIds`()):
    - 0 = NONE (not enough healthy equipment available for current/desired mode)
    - 1 = REDUCED (min <= number required units < max; for all equipment types)
    - 2 = FULL (number of required units = maximum; for all equipment type)
- `EquipmentStatus.readinessCurrent`

  `EquipmentStatus.readinessDesired`
  - Information about activation progress of units. If required equipment set for current/desired AOCS mode is ready, under activation or none (in case there is no available equipment set anymore)
  - Possible values (from `defActivationStatusIds()`)
    - 0 = NONE (not enough healthy equipment available for current/desired mode)
    - 1 = INPROG (activation is in progress)
    - 2 = READY (activation is over, units are expected to be available)

The items `UnitsStatusExpected` and `EquipmentStatus.availabilityCurrent` are key element for the FDIR. The first is the main source for validity information for the activation of unit focussed FDIR monitoring (on power level, communication level and operational level) and the second one in the indicator if the current AOCS mode cannot be maintained because of a lack of healthy units. In this case usually a system reconfiguration is required (e.g. by switching to the B branch of the avionic chain).

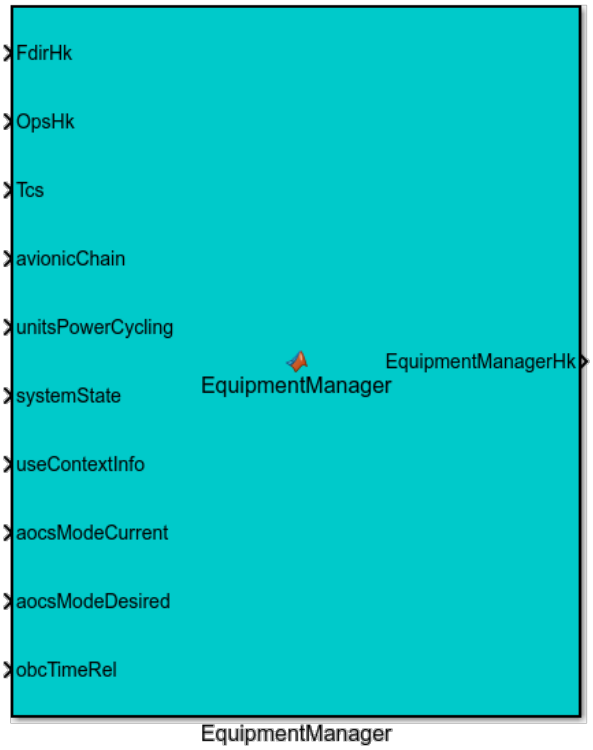


Figure 4-12: Equipment Manager.

4.6.5.2.1 Parameterization

The Equipment Manager can be configured with the set of parameters explained hereafter. They can be set in the `paramSystem.m` file in the corresponding section:

► Equipment Manager Sample Time: normally the sampleTime of the System Module.

```
% Sample time of Equipment Manager
sampleTime = systemSampleTime;
```

► Units Activation Delays

```
% Default delays for expected unit status (used during unit activation
sequence) [s]
UnitsActivationDelays.(<eqptShortName>)(unitId).off2pwr = 10;
UnitsActivationDelays.(<eqptShortName>)(unitId).pwr2com = 20;
UnitsActivationDelays.(<eqptShortName>)(unitId).com2op  = 30;
```

► Configuration Table

The unit configuration tables define for each AOCS equipment type, each AOCS mode and each used avionic chain which combinations of units can be used.

```
% Unit configuration tables
ConfigurationTable.(<eqptShortName>).(<modeName>).(<avionicChainName>)
= [0 1 1
   1 0 1
   1 1 0];
```

If one wants to define that in AOCS Mode “ASM” on avionic chain “A” any configuration of 2-out-of-3 magnetometers is sufficient, this would look like (column number equals configuration, row number equal unit ID):

```
% Specific unit configuration table for Mag, ASM, A
ConfigurationTable.Mag.ASM.A = [1 0 1
                                0 1 1
                                1 1 0];
```

In this case the first configuration (row 1) consisting of Mag1 and Mag3 would be activated for the current AOCS Mode if Mag1 and Mag3 are available, i.e. healthy. Otherwise the equipment manager would activate the first available configuration (top to bottom) or report “`EquipmentStatus. availabilityCurrent = NONE`” in case there is no valid configuration.

In order to ease the generation of the configuration table, the function “`generateUnit ConfigurationTable()`” can be used. The first input argument is the selection priority of the units (in the example below unit3 first, unit2 next, unit1 last), the minimum and maximum number of required units (here 1 and 2) and the last argument can be used to invert the given priorities (e.g. to use as “opposite” as possible configurations for avionic chain A and B). The input

```
generateUnitConfigurationTable([1 2 3], [1 2], false)
```

would result in the following table:

```
ans = [0 1 1
       1 0 1
       1 1 0
       0 0 1
       0 1 0
       1 0 0]
```

Whereas the output with “invertPriorities = true” would be:

```
ans = [1 1 0
       1 0 1
       0 1 1
       1 0 0
       0 1 0
       0 0 1]
```

4.6.5.2.2 Initial Conditions

In addition to the parameterization the Equipment Manager offers the initial conditions discussed hereafter. Like the parameterization, they can also be set in the `paramSystem.m` file in the corresponding section:

► Initial Unit Commands (Current & Pending)

```
% Initial unit on/off commands (current and pending)
UnitsCommandCurrent0.(<eqptShortName>)(unitId) = DefUnitCmdIds.ON;
UnitsCommandPending0.(<eqptShortName>)(unitId) = DefUnitCmdIds.ON;
```

► Initial Unit Health

```
% Initial unit health status
UnitsHealth0.(<eqptShortName>)(unitId) = 2;
```

► Initial Unit Status Expected

```
% Initial expected unit status
UnitsStatusExpected0.(<eqptShortName>)(unitId).isPwr  = 0;
UnitsStatusExpected0.(<eqptShortName>)(unitId).isCom  = 0;
UnitsStatusExpected0.(<eqptShortName>)(unitId).isOp   = 0;
UnitsStatusExpected0.(<eqptShortName>)(unitId).isUsbl = 0;
```

► Initial Unit On Time

```
% Specific initial "unit on" time (time since switch-on, used in unit
activation sequence) [s]
UnitsOnTime0.(<eqptShortName>)(unitId) = NaN;
```

4.6.5.3 AOCS Mode Manager

The AOCS Mode Manager (`AocsModeManager`, see Figure 4-13) is in charge of handling AOCS mode transitions. Its basic task is to receive mode change requests from FDIR, OPS or via telecommand, and to indicate the desired change of AOCS mode to the equipment manager. The normal mode change sequence is the following:

1. The AOCS Mode Manager receives a mode change request and sets its output “aocsModeDesired” to the new mode.
2. This is recognized by the Equipment Manager, which activates the equipment for the new mode and reports first “`EquipmentStatus.readinessDesired = INPROG`”, and at successful completion “`EquipmentStatus.readinessDesired = READY`”.
3. Once the required equipment is ready, the AOCS Mode Manager sets the current mode to the desired one and reset the mode duration timer.
4. The Equipment Manager recognizes the change of current mode and switches off all units which are not required anymore.

In addition the AOCS mode manager handles the setup of the intial AOCS mode at system (re)start. In this case the “initialAocsMode” contained in the current system configuration (as provided by the System Configuration Manager) is used as desired mode.

In case the “initialAocsMode” is set to “SAME” (i.e. same as before the restart), there are two options: If the “useContextInfo” of the current system configuration is either “RAM” or “SGM”

(safeguard memory), the desiredAocsMode is set to the last currentAocsMode from before the restart. In case the “useContextInfo” is “NONE”, the initial AOCS mode is determined by the parameter `defaultInitialAocsMode`.

In all cases: if the AOCS mode to change to is given only as main mode ID, then the entry submode of this main mode is looked up from the parameter table `entrySubModes`.

The following items are contained in the components output `AocsModeManagerHk:`

- `aocsModeCurrent:` ID of current AOCS mode (i.e. main + submode)
  - Possible values: all defined in `defAocsModeIds()`
- `aocsModeDesired:` ID of desired AOCS mode (i.e. main + submode)
  - Possible values: all defined in `defAocsModeIds()`
- `aocsMainModeCurrent:` ID of current AOCS main mode
  - Possible values: all main modes defined in `defAocsModeIds()`
- `aocsMainModeDesired:` ID of desired AOCS main mode
  - Possible values: all main modes defined in `defAocsModeIds()`
- `modeDuration:` Time since entry into the current AOCS mode in seconds.

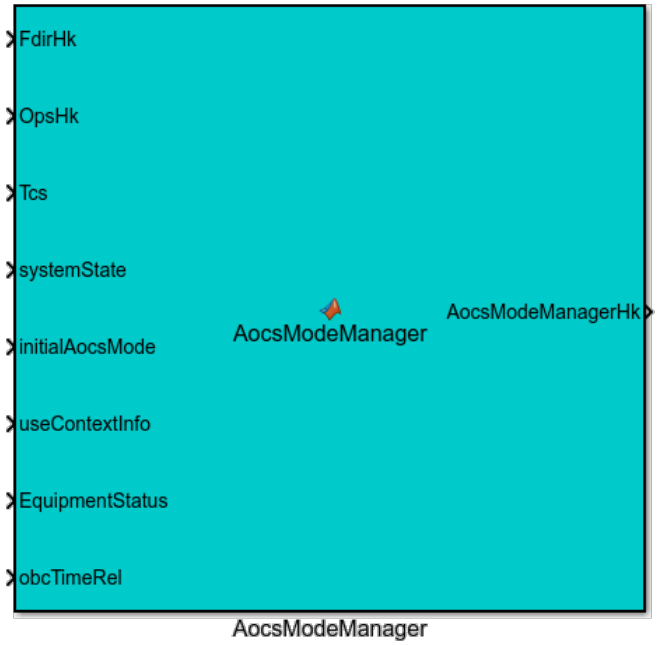


Figure 4-13: AOCS Mode Manager.

4.6.5.3.1 Parameterization

The AOCS Mode Manager can be configured with the set of parameters explained hereafter. They can be set in the `paramSystem.m` file in the corresponding section.

► AOCS Mode Manager Sample Time: normally the sampleTime of the System Module.

```
% Sample time of AOCS Mode Manager
sampleTime = systemSampleTime;
```

► Default AOCS Mode in case use of context info is not allowed, but no explicit mode is defined in the system configuration.

```
% If initial AOCS mode from SystemConfigManager after reboot is
% "same" and no context info is available, this submode is used.
defaultInitialAocsMode = DefAocsModeIds.ASM_RD;
```

► Entry submodes of all main modes

```
% Entry submode for each main mode
entrySubModes = [...
   DefAocsModeIds.OFF  DefAocsModeIds.OFF_OFF
   DefAocsModeIds.SBM, DefAocsModeIds.SBM_CO
   DefAocsModeIds.ASM, DefAocsModeIds.ASM_RD
   DefAocsModeIds.NOM, DefAocsModeIds.NOM_ACQ
   DefAocsModeIds.OCM, DefAocsModeIds.OCM_ACQ];
```

4.6.5.3.2 Initial Conditions

In addition to the parameterization the AOCS Mode Manager offers the initial conditions discussed hereafter. Like the parameterization, they can also be set in the `paramSystem.m` file in the corresponding section:

► Initial current and desired AOCS mode (at start of simulation)

```
% Initial AOCS Mode (current and desired)
aocsModeCurrent0 = DefAocsModeIds.SBM_CO;
aocsModeDesired0 = DefAocsModeIds.ASM_RD;
```

4.6.6 Command

The Command module consists of a single component “SendCommands”. It is designed to send telecommands (TCs) to the System Module.

The parameterization of the component realizes a time line of TC events which is defined in `paramCommand.m` (in …`\data\param\`).

Figure 4-14: Only component of Command module.

4.6.6.1 Possible Telecommands

The following types of TCs are currently implemented:

- System Reboot
  - Example: the following parameterization executes at simulation time 1111s an AOCS Mode transition to ASM_RD:

```
iTc = 1;
UserCommand(iTc).time  = 1111;
UserCommand(iTc).type  = DefTcIds.PERFORM_REBOOT;
UserCommand(iTc).value = 1; % No meaning here, but != 0
```

- AOCS Mode Transition
  - Example: the following parameterization executes at simulation time 2222s an AOCS Mode transition to ASM_RD:

```
iTc = 1;
UserCommand(iTc).time  = 2222;
UserCommand(iTc).type  = DefTcIds.GOTO_AOCS_MODE;
UserCommand(iTc).value = DefAocsModeIds.ASM_RD;
```

- Stop Simulation
  - Example: the following parameterization executes at simulation time 2222s an AOCS Mode transition to ASM_RD:

```
iTc = 1;
UserCommand(iTc).time  = 2222;
UserCommand(iTc).type  = DefTcIds. STOP_SIMULATION;
UserCommand(iTc).value = 1; % No meaning here, but != 0
```

4.6.7 Fault Injection

The FaultInjection module consists of a single component “InjectFaults”. It is used to provide the the information of present faults to other modules. Faults can have an effect on equipment, the AOCS algorithms and the System Module.

The parameterization of the component realizes a time line of fault events which is defined in `paramFaultInjection.m` (in …`\data\param\`).

The output bus of the component only carries the basic information about the fault events (i.e. its ID, presence and persistency). All related model parameters are defined in the

parameterization file of the respective module which is affected by the fault. Please refer to section 4.9 for a detailed description on how to define and parameterize faults.

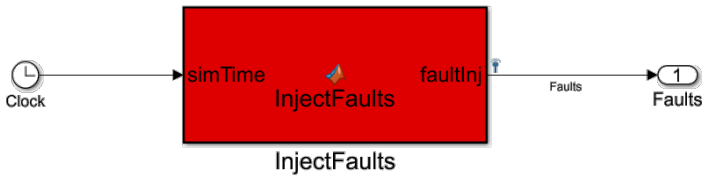


Figure 4-15: Only component of FaultInjection module.

The parameterization of the fault injection itself is straightforward. The main required information is the target module, the target component, the target unit ID and the type of fault. In addition the presence flag tells whether to inject (1) or eject (0) the fault and if the fault shall be present after a restart of the unit (currently only used for AOCS equipment faults).

```
idx = 1;
UserFaultInjection(idx).module      = 'Equipment';
UserFaultInjection(idx).component   = 'Mag';
UserFaultInjection(idx).unit        = 2;
UserFaultInjection(idx).time        = 1000;
UserFaultInjection(idx).presence    = 1;
UserFaultInjection(idx).persistency = 0;
UserFaultInjection(idx).type        = 'STALEDATA';
```

4.7 Developing own Equipment Models

This section describes how new equipments can be created and how existing equipments can be modified. As being part of the “Generic Equipment Model” (GEM), all equipments need to share a certain set of parameters, behavioural properties and interfaces. This concept simplifies the common initialization of all “specific equipment models” (SEM) for the different sensor and actuator models, i.e. the setup of Simulink buses, block parameter structures as well as the noise and state vector generation.

Apart from this generic part which basically forms the “skeleton” of the equipment models, the user is free in extending the I/O properties (additional housekeeping and (auxiliary) status outputs, command inputs to actuators, states) and is entirely free in defining the model “core” of an equipment including various signal manipulations and faults to be applied to an equipment.

Note, that the generic part of SEM will be automatically interfaced e.g. with the System module, observing e.g. the operational state. The specific part of a SEM instead needs a specific counterpart which mostly is related to parts of the AocsAlgo module.

Generally, the following “types” of performance models are distinguished. These types are also reflected in the structure of the equipment parameterization.

- **Ideal Performance Model** (“Config” part of equipment parameterization structure): summarizes all computations and transformations that are required to obtain ideal measurements of quantities provided by the environmental module or to perform ideal actuation. Ideal performance is not degraded in any sense, but takes following basic constraints into account: FoV, eclipse, i.e. only physical constraints apply.

  *Example***:** the magnetic field measurement of a magnetometer returns the exact magnetic field provided by the magnetic field model for the current position and orientation of the magnetometer.
- **Nominal Performance Model** (“SigMa” part of equipment parameterization structure): summarizes all kind of signal manipulations between inputs and outputs of a SEM core function with the goal to obtain a realistic behaviour of the modelled unit (model a unit according to data sheet).

  *Example***:** bias, noise and scale-factor on a magnetic field measurement of a magnetometer in the absence of any fault.
- **Fault Performance Model** (“Fault” part of equipment parameterization structure)**:** summarizes all kind of signal manipulations between inputs and outputs of a SEM core with the goal to obtain a realistic behaviour of the modelled unit under the presence of a specific fault.

  *Example***:** increased bias and noise on a magnetic field measurement of a magnetometer in case the sensor is overheated.

4.7.1 Create new Equipments

Creating an entirely new equipment model requires several steps which are further described in the following subsections:

- Equipment Definition: definition of I/O properties and unique identifiers (UIDs, unique within one equipment only) for nominal signal manipulations
- Equipment Faults: definition of faults to be applied to the units
- Equipment Parameterization: generation of a parameter file to specify parameter values for the (ideal, nominal and fault performance) model and implement the selection of an active number of units for the equipment
- Equipment Initialization: generation of an initialization file to (optionally) apply checks and conversions on the user parameters
- Equipment Core Model: creation of a file for the equipment model core algorithm and add the core algorithm call to the GEM
- Equipment Plot Configuration

Note that only the first two steps need to be carried out manually. The remaining steps can be mainly automated using the function `createEquipment('<EqShortName>',` which generates all required files and prepares the parameter structure and core algorithm skeleton (which then needs to be completed by the user). Here, `<EqShortName>` is a string for the

abbreviated equipment name provided in the equipment definition file (see 0) of the project (make sure that `gafeProjectStartup` is called from the project folder before).

When using `createEquipment`, it is only mandatory to introduce the new equipment in `defEquipments.m` (as described in the next section) beforehand and faults can optionally be defined in advance in `defEquipmentFaults.m`, see section 4.7.1.2).

The more information can be provided at this point (about desired housekeeping and states outputs, signal manipulations etc.), the less manual modifications are required afterwards to complete the function skeletons to a complete model. For the latter task, please also refer to section 4.7.2 about modifying existing equipment models.

The reaction wheel model (present in the GAFE library) serves as example throughout the following sections.

4.7.1.1 Equipment Definition

Defining the fundamental properties of an equipment is realized in the definition function `defEquipments.m` located in …`\data\def\properties\`. No parameter values are specified in this definition file. For each equipment type, a structure as shown in the template below is required, thus edit the file and fill the requested information:

```
%%
% MyEquipment
%=========================================================================


% id = id + 1;        % Always keep this line of code at the beginning

% Def.<myShortName>.id                    = id;
% Def.<myShortName>.shortName             = 'myShortName';
% Def.<myShortName>.longName              = 'myLongName';
% Def.<myShortName>.displayName           = 'myDisplayName';
%-------------------------------------------------------------------------
% Def.<myShortName>.Uid(1).name           = 'myUidName';
% Def.<myShortName>.Uid(1).size           = <mySize>;
% Def.<myShortName>.Uid(1).type           = 'mySigMaType';
% Def.<myShortName>.Uid(1).output         = myOutputType;
% Def.<myShortName>.Uid(1).isActCmd       = myActFlag
% Def.<myShortName>.Uid(1).nomSigMa       = {<mySigMaList>}
%-------------------------------------------------------------------------
% Def.<myShortName>.Uid(2).name           = 'myUidName2';
% Def.<myShortName>.Uid(2).size           = <mySize2>;
% Def.<myShortName>.Uid(2).type           = 'mySigMaType2';
%                 :
%                 :
```

The various placeholders are explained below. The first “block” is related to definitions for the overall equipment.

- id:

  The ID is used to internally identify the equipment model (e.g. to automatically generate noise seeds and fault injection signals). Please keep the provided "`id = id + 1`" scheme.

- myShortName:

  This is a string representing an abbreviation for the equipment used for the automatic initialization and generation of buses and structures.

```
Def.Rw.shortName              = 'Rw';
```

- myLongName:

  A string (no spaces) representing a long name for the equipment used for the automatic initialization (the respective init/paramfiles for this equipment must be named accordingly, i.e. `param<myLongName>.m` and `init<myLongName>.m`.

```
Def.Rw.longName               = 'ReactionWheel';
```

- myDisplayName:

  A string representing the name for the equipment used in error/warning messages during the initialization of this equipment; different to `<myLongName>`, spaces can be used in the string.

```
Def.Rw.displayName              = 'Reaction Wheel';
```

All subsequent "blocks" are related to the definition of UIDs for the equipment. An arbitrary number of these blocks can be defined dependent on the needs for the model. The array indexing must be kept consecutive.

- myUidName:

  A string for the UID which is used to define important properties of an equipment core:

  - output assignments
  - variables which are subject to nominal signal manipulations

  Together with further information from the parameters below this allows automated consistency check and parameter preparation during the initialization.

- mySize:

  The size associated with the UID, e.g. [3,1] for a 3x1 vector.

- mySigMaType:

  The "signal" type associated with the UID, see `defUidTypes.m`. This is required as "usual" vectors, unit vectors and quaternions need to be treated differently in the signal manipulation.

  Note: a scalar is a special case of a "usual" vector.

- myOutputType:

  The scalar output type ID associated with the UID as defined in defEquipmentOutputTypes.m

  - housekeeping output available to AOCS, System, and FDIR
  - auxiliary status outputs for scenario analysis/inspection
  - state output used for dynamical integration of equipment states
  - no output, i.e. UIDs only to be used internally in the core algorithm

- myActFlag:

  (optional) Boolean variable indicating if this UID relates to an actuation command of an actuator (used to automatically generate the bus for unit commands from the AocsAlgo module). If this field is missing, "false" is assumed as default.

- mySigMaList:

  A list of IDs that defines what kind of signal manipulation apply to this UID during **nominal** operation (“nominal performance model”) of this equipment. A complete list of IDs is provided in `defSigMaTypes.m`, see also section 4.7.3.2.

  Notes:

  - Faults associated with the UIDs defined here which replace the nominal behaviour are explicitly defined in `defEquipmentFaults.m`, see also section 4.9.1.1.
  - If no manipulation shall be applied, simply use an empty cell, i.e. "{}"

Examples:

```
Def.Rw.Uid(1).name            = 'rwRateMeas';
Def.Rw.Uid(1).size            = [1 1];
Def.Rw.Uid(1).type            = DefUidType.VEC;
Def.Rw.Uid(1).output          = DefOutputs.HK;
Def.Rw.Uid(1).nomSigMa        = {...
   DefSigMa.GAUSSMK.str ...
   DefSigMa.QUANT.str
```

This defines the UID ‘`rwRateMeas`’ for the reaction wheel rate as housekeeping output of length 1. A Gauss-Markov noise model and a quantization is applied to this measurement.

```
Def.Rw.Uid(2).name            = 'actTrqCmd';
Def.Rw.Uid(2).size            = [1 1];
Def.Rw.Uid(2).type            = DefUidType.VEC;
Def.Rw.Uid(2).output          = DefOutputs.STATUS;
Def.Rw.Uid(2).isActCmd        = true;
Def.Rw.Uid(2).nomSigMa        = {...
   DefSigMa.SATURATION.str};
```

This defines the UID ‘actTrqCmd’ for the reaction wheel actuation torque command as command input and also as auxiliary status output of length 1. A saturation is further applied to the command.

```
Def.Rw.Uid(3).name            = 'rwRate';
Def.Rw.Uid(3).size            = [1 1];
Def.Rw.Uid(3).type            = DefUidType.VEC;
Def.Rw.Uid(3).output          = DefOutputs.STATE;
Def.Rw.Uid(3).nomSigMa        = {};
```

This defines the UID ‘rwRate’ for the actual reaction wheel rate as state of as length 1. For states, an input (the state itself) and an output (the state derivative to be fed to the integrator in the GEM) are automatically created.

```
Def.Rw.Uid(5).name            = 'coulombTorque';
Def.Rw.Uid(5).size            = [1 1];
Def.Rw.Uid(5).type            = DefUidType.VEC;
Def.Rw.Uid(5).output          = DefOutputs.NONE;
Def.Rw.Uid(5).nomSigMa        = {};
```

This defines the UID ‘coulombTorque’ for the Coulomb torque coefficient of length 1. This UID is neither assigned to an output nor underlying any signal manipulations from the nominal performance model. Its setup in this manner is however still required in case a defined fault has an impact in this UID.

4.7.1.2 Equipment Faults

Defining the fundamental properties of faults to be applied to an equipment is realized in the definition function `defEquipmentFaults.m` located in …\data\def\properties\. No parameter values are specified in this definition file. Faults can affect the entire unit (i.e. operational states or its transitions) or the manipulation of UIDs. More details about defining faults for an equipment are provided in section 4.9.1.

When the function `createEquipment` function is used to create a new equipment <Equipment> the following set of “default” faults is automatically appended in the definition file for this equipment (also in case other user-defined faults have already been manually defined before for the new equipment):

```
% Generic transition fault
Fault.<EqShortName>.TRANSITION.unitFault   = [Id.TRANS];
```

A generic fault which allows to manipulate the transition between operational states of the equipment.

```
% Complete failure
Fault.<EqShortName>.COMPLETE.unitFault     = [Id.NOPOW, Id.NOCOM];
Fault.<EqShortName>.COMPLETE.<actCmdUid>   = [Id.NOVAL];
Fault.<EqShortName>.COMPLETE.<HkUid1>      = [Id.NOVAL];
Fault.<EqShortName>.COMPLETE.<HkUid2>      = [Id.NOVAL];
                  :
```

A complete failure of an equipment defined by disabling power and communication of the equipment and setting all housekeeping data (and actuation commands if present) to NaN.

```
 % Communication failure
Fault.<EqShortName>.COM.unitFault           = [Id.NOCOM];
Fault.<EqShortName>.COM.<actCmdUid>         = [Id.NOVAL];
Fault.<EqShortName>.COM.<HkUid1>            = [Id.NOVAL];
Fault.<EqShortName>.COM.<HkUid2>            = [Id.NOVAL];
                  :
```

A communication failure of an equipment defined by disabling communication of the equipment and setting all housekeeping values to NaN while maintaining the power status.

```
% Command failure
Fault.<EqShortName>.COM.<actCmdUid>         = [Id.NOVAL];
```

A command failure of an equipment defined by setting the actuation command to NaN (only if an actuation command is defined for unit).

Note:

Without using the `createEquipment` function (or after having used it for initial creation), it is always possible to add or modify the fault definition. If any elementary faults added require user parameters (see section 4.9.1.1), the respective parts of the parameter structure then have to be manually added to the parameter file of the equipment, i.e. to `param<MyEquipment>.m`, see section 4.7.2.3.

4.7.1.3 Equipment Parameterization

4.7.1.3.1 param[MyEquipment].m

To specify all parameter values for the ideal, nominal and fault performance model of a new equipment <MyEquipment>, a parameter file named `param<EqLongName>.m` needs to be created in …`\data\param\SEM\`.

The most convenient way to create the file is to use the function `createEquipment`. This creates the file itself, fills it with the entire parameter structure required (derived from the definition file of the equipment and from the parameters common to all equipments from the GEM) and assigns “neutral” parameter values to all parameters of proper size. The only task for the user is then to adapt the parameter values to the desired ones.

Examples for automatically generated entries based on the definition in `defEquipments.m`:

- Nominal saturation is applied to the UID `‘actTrqCmd’`: limits are required

```
UserRw.Default.SigMa.actTrqCmd.Saturation.maxValue     = Inf;
UserRw.Default.SigMa.actTrqCmd.Saturation.minValue     = -Inf;
```

- “actTrqCmd” is defined as status output: initial value must be present

```
UserRw.Default.InitStatus.actTrqCmd                      = 0;
```

Similarly, automatically generated entries also exist based on the fault definition in `defEquipmentFaults.m`, e.g.:

- Fault bias and scale error defined by a fault named `'WRONTRQ'` require values:

```
UserRw.Default.Fault.WRONGTRQ.actTrqCmd.Bias.value        = 0;
UserRw.Default.Fault.WRONGTRQ.actTrqCmd.SymScaleErr.value = 0;
```

Note:

To modify the parameter structure or to manually create the entire parameterization file, please refer to the detailed description of the parameter file structure in section 4.7.2.3.

4.7.1.3.2 paramEquipment.m

The parameter function `paramEquipment.m` controls the number of units for each equipment which shall be present in the simulation. When creating a new equipment, the selection has to be extended to account for the new <MyEquipment> by adding the following lines:

```
% <MyEquipment>
UserEquipment.<MyEquipment>.numUnits                    = value;
```

Using the function `createEquipment`, this is treated automatically.

4.7.1.4 Equipment Initialization

The initialization of all parameters common to all equipment is treated in the generic function `initEquipment` which requires no user modifications at all. However, for the new <MyEquipment>, an initialization file named `init<EqLongName>.m` still needs to be created in …`\data\param\SEM\`. This is necessary to allow the initialization of any model specific parameters which are not covered by the GEM.

When creating the file manually, copy an init-file from an existing equipment model and then further adapt its content (the more convenient way to create the file is however to use the function `createEquipment` again). This initialization file contains three editable sections which are described below.

Equipment specific parts of the default structure

The subfunction "`addEquipmentSpecificDefaultStruct`" is used to define equipment specific parameters which are expected to be present in the `Config` sub-structure of `param<MyEquipment>`. Only specific variables defined here are later present in the simulation parameter structure.

This approach was chosen for two reasons. First, to be able to detect any missing parameter during initialization (due to typos, forgotten settings etc.) and second, to allow specifying fixed model parameters which shall usually not be modified by a "later user" via parameterization (e.g. a reference epoch).

It is only required to define the parameter names, providing explicit values is not mandatory apart for the reason mentioned above:

```
DefEquipment.Config.<mySpecVar> = [];
```

Example:

```
DefThisEquipment.Config.inertia         = [];
DefThisEquipment.Config.coulombTorque   = [];
DefThisEquipment.Config.viscFrctGain    = [];
DefThisEquipment.Config.viscFrctExp     = [];
DefThisEquipment.Config.spinAxis_U      = [];
```

This defines all required parameters which are specific to the reaction wheel model behaviour only and which cannot be covered via the GEM.

Checks and conversions of user parameters

The content of this section is closely related to the parameters defined in the previous section. It can be used to check the consistency of the specific parameter values provided in `param<MyEquipment>` or to convert them to units required internally by the model (e.g. it might be convenient to specify a wheel speed in rpm while the model core model itself requires the respective SI unit rad/s).

Applying checks and conversion to the specific user parameters is completely optional.

Example:

Check that the provided inertia value is a positive scalar and throws an error during the initialization if this is not the case.

```
% Wheel inertia
assert(isscalar(ThisEquipmentPar.Config(i).inertia) && ...
   ThisEquipmentPar.Config(i).inertia > 0, ...
   ['Inertia of ',lower(eqDisplayName),' (unit ', num2str(i),...
   ') must be a scalar value > 0.']);
```

Initial dynamics contribution from the unit

Any equipment which has an actuation command as inputs (determined via the presence of an `isActCmd` flag for any UID in `defEquipments.m`) can contribute to the overall S/C dynamics.

If this is the case, the initial contribution to the unit dynamics at simulation start (force, torque and angular momentum) can be specified by assigning:

```
ThisEquipmentPar.InitUnitsDyn(i).frc_B    = [3x1 vector];
ThisEquipmentPar.InitUnitsDyn(i).trq_B    = [3x1 vector];
ThisEquipmentPar.InitUnitsDyn(i).angMom_B = [3x1 vector];
```

These contributions depends usually on other model parameters (initial values, configuration parameters) and cannot be generated automatically. If not specified explicitly zero vectors are assumed.

Example:

```
% RW Torque
ThisEquipmentPar.InitUnitsDyn(i).trq_B = -
ThisEquipmentPar.InitStatus(i).rwTrq_B;

% Rw Angular momentum
angMom_U = ...
   ThisEquipmentPar.Config(i).inertia ...
   * ThisEquipmentPar.InitState(i).Value.rwRate ...
   * ThisEquipmentPar.Config(i).spinAxis_U;

ThisEquipmentPar.InitUnitsDyn(i).angMom_B = ...
   ThisEquipmentPar.Config(i).att_dcm_UB' *  angMom_U;
```

4.7.1.5 Equipment Core Model

4.7.1.5.1 Model Core Algorithm

To define the actual runtime model for the new <MyEquipment>, a file named `<eqShortName>CoreAlgorithm.m` needs to be created in …`\src\Equipment\SEM\`.

The most convenient way to generate this file is using the function `createEquipment.m`. It automatically creates the correct syntax for the function call, initializes required values and adds signal manipulation calls based on the equipment definition in `defEquipments.m`. Further, it provides a skeleton for assigning housekeeping outputs dependent on the operational states of the units.

It is also possible to create the core algorithm file manually without using `createEquipment.m`. In this case, please refer to section 4.7.2.4 for a more detailed description of the core algorithm design.

4.7.1.5.2 Calling the SEM Core from the GEM

To enable the call of the specific equipment model core, the switch case expression in the generic equipment model must be extended. This extension is automatically realized using the function `createEquipment.m`.

It is also possible to implement the function call manually in the GEM. In this case, please refer to section 4.7.2.4 for a more detailed description.

4.7.1.6 Equipment Plot Data Configuration

To define the plot configuration (labels, titles, axis ranges, ticks, etc.) for the new <MyEquipment>, a file named `<eqLongName>_cfg.m` needs to be created in …`\data\plotCfg\`.

The most convenient way to generate this file is using the function `createEquipment.m`. It automatically creates a template for all status and housekeeping outputs related to the defined equipment.

See also section 4.4.6.1 for more information about plotting and plot configuration.

4.7.2 Modifying existing Equipments

This section describes how existing equipment models can be modified. Modifications can be related to creating additional inputs/outputs, introducing new parameters and changing the “ideal”, “nominal” and “fault“ performance model of an equipment (both in terms of signal manipulation parameters applied and the core model “behaviour” itself). The following section briefly describes the required modifications for “typical” cases. Subsequent chapters describe in more detail all modifications which can be applied to individual files.

4.7.2.1 Quick-Guide

If you want to **rename a (status, housekeeping or state) output** of an <Equipment>, the following files need to edited or checked:

- **defEquipments.m**

  Search for the entries of the respective equipment and rename the affected UID name:

```
Def.<EqShortName>.Uid(i).name    = '<newName>';
```

- **defEquipmentFaults.m**

  Check if the respective UID is present for any fault defined for the equipment and replace it with <newName>.

- **param<eqLongName>.m**

  Check where the respective UID is present and simply replace it with <newName> .

- **<eqShortName>CoreAlgorithm.m**

  Check where the respective UID is present and simply replace it with <newName>.

- **init<eqLongName>.m**

  Check if the respective UID is present and simply replace it with <newName>.

- **<EqLongName>_cfg.m**

  Check if the respective UID is present and simply replace it with <newName>.

If you want to **add a (status, housekeeping or state) output** of an <Equipment>, the following files need to edited:

- **defEquipments.m**

  Introduce and configure a new UID for the output by appending it to the existing UIDs

```
Def.<EqShortName>.Uid(end+1).name           = 'myUidName';
Def.<EqShortName>.Uid(end+1).size           = <mySize>;
Def.<EqShortName>.Uid(end+1).type           = 'mySigMaType';
Def.<EqShortName>.Uid(end+1).output         = myOutputType;
Def.<EqShortName>.Uid(end+1).nomSigMa       = {<mySigMaList>};
```

  See section 4.7.1.1 for more details on the parameters.

- **defEquipmentFaults.m**

  Optionally, let the new UID be affected by faults (see later in this section)

- **param<eqLongName>.m**

  Depending on the type of the output, define:

  - the initial status output in part B)4)i)
  - the initial housekeeping output in part B)4)ii)
  - the initial state and state limits in part B)4)iv)

  If the output is subject to one or more signal manipulations, define the respective manipulation parameters in part A)2)ii).

  If the output is subject to one or more faults, define the respective fault parameters in part A)3)ii).

- **<eqShortName>CoreAlgorithm.m**
  - Status output:

    Add the new status output to the initialisation structure at the beginning and to the status output assignment at the end of the file.
  - Housekeeping output:

    Add the new housekeeping output to the initialisation structure at the beginning of the file and as input to the HkMem/HkMemUpdate functions in the core part of the file
  - State output:

    Add the new state output to the status initialisation structure (state outputs are always fed to a status output) at the beginning and to the status output assignment at the end of the file.

    Further, retrieve the actual state part from the input state vector:

    ```
    stateIndex = UnitConfig.StateIndex.<UidName>;
    <UidName> = makeColumnVec(stateVec(stateIndex);
    ```

    And set the respective function output for the state derivative vector

    ```
    stateDxVec(stateIndex) = <sth. of size of the new output>
    ```

  If the state input is the first state input of the equipment, the function call of the core algorithm also needs to be adapted according to feed in/out the states and its derivatives:

  ```
  [ …, stateDxVec] =  <eqShortName>CoreAlgorithm(…,stateVec)
  ```

- **genericEquipmentModel.m**

Editing this file is only necessary if the new output is the first state output defined for the equipment. In this case, the function call for the equipment has to be extended as follows to feed the states and derivatives from/to the external integrator:

```
[ …, ...
 unitsDxVec(Config.(nameEq{idxEq})(idxUnit).StateIndex.global)]
 = <eqShortName>CoreAlgorithm(...
   …, ...
 unitsStateVec(Config.(nameEq{idxEq})(idxUnit).StateIndex.global);
```

- **init<eqLongName>.m**

  Editing this file is only required, if the new output shall affect the spacecraft dynamics. In this case, the initial contribution of the output to body force, torque and angular momentum can be defined at the end of the file.

- **<EqLongName>_cfg.m**

  Optionally, introduce plot configuration parameters (labels, title, ticks) for the new output.

If you want to **add a persistent variable** (i.e. an “internal state”) to an <Equipment> core algorithm, the following files need to edited:

- **<eqShortName>CoreAlgorithm.m**

  Add the desired variable <myStateVar> as input and output to the function call (this is required to be compatible with multiple instances of an equipment.

  ```
  [ …, <myStateVar>] = ...
  <eqShortName>CoreAlgorithm(…, <myStateVar>);
  ```

- **genericEquipmentModel.m**

  Add the desired variable `<myStateVar>` to the list of persistent variables at the beginning of the function together with a persistent variable to check for the first call `isFirst<myStateVar>Call`.

  Initialize the persistent variable in the switch-case expression for the <Equipment> right before the core algorithm call in the enclosed in a check for the first call. The size of the variable must take into account a varying number of units.

  In case of a scalar “numeric” state for each unit, for instance:

  ```
  % Initialize persistent variable
    if isempty(isFirst<myStateVar>Call)
       <myStateVar> = zeros(thisEqNumUnits,1);
       isFirst<myStateVar> Call = false;
    end
  ```

In case of a state defined by a structure (two fields, first property is scalar, second property is a 3x1 vector), for instance:

```
% Initialize persistent variable
  if isempty(isFirst<myStateVar>Call)
    <myStateVar> = struct( ...
      '<fieldName1>', repmat({0}, thisEqNumUnits,1),...
      '<fieldName2>', repmat({[0 0 0]', thisEqNumUnits,1));
     isFirstGnsrCall = false;
  end
```

Note that the structure field cannot be in sequence, but only in "one shot" due to compilation constraints.

Finally, add the <myStateVar> part for the current unit as input and output to the function call of the core algorithm:

```
[ …, <myStateVar>(idxUnit)] = ...
<eqShortName>CoreAlgorithm(…, <myStateVar>(idxUnit);
```

If you want to **add/rename a configuration parameter** (`.Config.<param>`)of an equipment <Equipment>, the following files need to edited:

- **param<eqLongName>.m**

  For renaming, simply check where the respective `<param>` is present in part and simply replace it with `<newParam>` in part A)1)ii).

  To add a new parameter, simply extend part A)1)ii) by

  ```
  User<EqShortName>.Default.Config.<newParam>  = <newParamValues>
  ```

  Names of common parameters in sections A)1)i) and A)2)i) may not be modified!

- **init<eqLongName>.m**

  For renaming, simply check where the respective `<param>` is present in the file and replace it with `<newParam>`.

  To add a new parameter, extend the definition list in the first subfunction `addEquipmentSpecificDefaultStruct` (the structure field can be left empty):

  ```
  DefThisEquipment.Config.<newParam> = [];
  ```

  If any checks to the new user parameter shall be applied during initialization, these checks can be placed in the subfunction `checkEquipmentSpecificParam`.

- **<eqShortName>CoreAlgorithm.m**

  For renaming, simply check where the respective `<param>` is present in the file and replace it with `<newParam>`.

  New parameters can be retrieved for use in the core algorithm from the input `UnitConfig.<newParam>`.

If you want to **modify a nominal signal manipulation of an existing UID** of an <Equipment>, the following files need to edited:

- **defEquipments.m**

  Modifying this file is only necessary, when a new manipulation type (see section 4.7.3.2) shall be added. In this case, extend the `.nomSigMa` field of the UID with the ID of the manipulation to be introduced. For instance, the modification in red below would introduce a bias to second UID of an equipment model:

```
Def.<eqShortName>.Uid(2).name          = '<uidName>';
Def.<eqShortName>.Uid(2).size          = [3 1];
Def.<eqShortName>.Uid(2).type          = DefUidType.VEC;
Def.<eqShortName>.Uid(2).output        = DefOutputs.HK;
Def.<eqShortName>.Uid(2).nomSigMa      = {...
   DefSigMa.BLWN.str  ...
   DefSigMa.GAUSSMK.str ...
   DefSigMa.BIAS.str};
```

- **param<eqLongName>.m**

  Generally, all manipulation parameter values can be modified in part A)2)ii) of the file. When introducing a new manipulation to a UID, the respective parameter list needs to be extended accordingly (see section 4.7.3.2 or `defSigMaTypes.m` which is located in the folder …`\data\def\properties\`.). For the example above:

```
User<EqShortName>.Default.SigMa.<uidName>.Bias.value = ...
[1,2,3]';
```

If you want to **add a new UID for signal manipulation** to an <Equipment> core model, the following files need to edited:

- **defEquipments.m**

  Add the following snippet below to the end of the definition of the equipment and fill the `.nomSigMa` field with the IDs of the manipulations (see section 4.7.3.2) to be applied to the UID (`DefOutputs.NONE` indicates that this UID is only used internally in the equipment model, but not related to an output in this case)

```
Def.<eqShortName>.Uid(<idx>).name          = '<uidName>';
Def.<eqShortName>.Uid(<idx>).size          = <mySize>;
Def.<eqShortName>.Uid(<idx>).type          = <myType>;
Def.<eqShortName>.Uid(<idx>).output        = DefOutputs.NONE;
Def.<eqShortName>.Uid(<idx>).nomSigMa      = {...
   DefSigMa.BIAS.str  ...
   DefSigMa.BLWN.str};
```

  In this example, a bias and band-limited white noise is applied to the UID.

- **param<eqLongName>.m**

Doc.No.: GAFE-UM-D7.5b
Issue: 2.1
Date: 17.08.2018

Extend the list of manipulation parameter values in part A)2)ii) of the file by the ones required for the introduced manipulations (see section 4.7.3.2 or `defSigMaTypes.m` which is located in the folder …`\data\def\properties\`.). For the example above:

```
User<EqShortName>.Default.SigMa.<uidName>.Bias.value = ...
   <a vector of values of <mySize> >;
User<EqShortName>.Default.SigMa.<uidName>.Blwn.std = ...
   <a vector of values of <mySize> >;
```

- **<eqShortName>CoreAlgorithm.m**

  Add the following snippet to the equipment core algorithm where the signal manipulation shall be applied:

```
[<actualSignal>, UnitNoiseData] = manipulateSignal(...
<nominalSignal>, ...
SigMaParams, ...
<previousSignal>, ...
sampleTime, ...
noiseVec, ...
UnitNoiseData, ...
'<uidName>', ...
performanceIndex);
```

Section 4.7.3.3.4 provides more information about the actual meaning of the various inputs and outputs.

If you want to **modify an existing fault** applicable to <Equipment>, the following files need to edited:

- **defEquipments.m**

  This file only needs to be modified if the modification of the fault introduces a new UID for the <Equipment>. In this case, the new UID needs to be introduced similar to the case of a new UID for signal manipulation described above. In this case, the `.nomSigMa` field can be left empty when used for fault manipulation only.

- **defEquipmentFaults.m**

  Extend the fault behaviour of an existing fault by introducing additional elementary fault type IDs (see section or 4.9.1.1 or `defElementaryFaults.m` which is located in the folder …`\data\def\properties\`) :

```
Fault.<eqShortName>.<FAULTNAME>.<existingUID> = <ID vector>;
```

  and/or by assigning new UIDs to the fault (together with elementary fault IDs):

```
Fault.<eqShortName>.<FAULTNAME>.<newUID> = <ID vector>;
```

As an artificial example, the red marked extension below introduces additional outliers to the measured APM angles, sets the commanded angles to NaN and introduces a faulty state transition of the unit for the fault POSMEASJUMP applicable to the antenna pointing mechanism.

```
Fault.Apm.POSMEASJUMP.apmAngMeas = [Id.BIAS, Id.OUTLIER];
Fault.Apm.POSMEASJUMP.apmRateCmd = [Id.NOVAL];
Fault.Apm.POSMEASJUMP.unitFault  = [Id.TRANS];
```

param<eqLongName>.m

The respective fault parameters for the UIDs need to be updated in part B)3)i) and B)3)ii) of the file (in case the elementary faults require further parameters, see section 4.9.1.1).

For the above example part B)3)i) needs the extension:

```
UserApm.Default.Fault.POSMEASJUMP.unitFault.Transition.paramSet
= <1x4 vector>; % see comments in param file for description
```

And part B)3)ii) needs the extension

```
UserApm.Default.Fault.POSMEASJUMP.ampAngMeas.Outlier.factor
= <2x1 vector>;
UserApm.Default.Fault.POSMEASJUMP.ampAngMeas.Outlier.occurence
= <2x1 vector>;
```

As a “no value” faults (Id.NOVAL) do not require any parameter values, nothing needs to be specified

- **<eqShortName>CoreAlgorithm.m**

  If a UID has been introduced for the modified fault which has not already a signal manipulation call in the core algorithm the following snippet has to be added at the desired location in the file:

```
[<actualSignal>, UnitNoiseData] = manipulateSignal(...
<nominalSignal>, ...
SigMaParams, ...
<previousSignal>, ...
sampleTime, ...
noiseVec, ...
UnitNoiseData, ...
‘<uidName>’, ...
performanceIndex);
```

Section 4.7.3.3.4 provides more information about the actual meaning of the various inputs and outputs.

If you want to **add a new fault** applicable to <Equipment>, the procedure is identical to the one described above for modifying an existing Fault, except that the new fault <NEWFAULTNAME> needs to be introduced in:

- **defEquipmentFaults.m**

```
Fault.<eqShortName>.<NEWFAULTNAME>.<UID1> = <ID vector>;
Fault.<eqShortName>.<NEWFAULTNAME>.<UID2> = <ID vector>;
...
```

As an artificial example, the following snippet could be used to introduce a fault “FLIP” that flips the sign of the commanded angles in the antenna pointing mechanism model:

```
Fault.Apm.FLIP.apmAngCmd = [Id.INVSIGN];
```

If you want to **add a new actuation command input** <newCmd> to an <Equipment>, the following files need to edited:

- **defEquipments.m**

  Extend the definition of an existing status output UID (or define a new status output UID as described above with all related modifications) by:

```
Def.<EqShortName>.Uid(i).isActCmd           = 'yes';
```

  Command inputs always need to be linked to a status output UID.

- **<eqShortName>CoreAlgorithm.m**

  This function call for the core algorithm only needs to be adapted if the introduced command is the first defined for this equipment. In this case, the function call has to be adapted by adding:

```
[ … ] =  <eqShortName>CoreAlgorithm(…,UnitCmd)
```

  It can be accessed in the function from `UnitCmd.newCmd`.

If you want to **let an <Equipment> affect the spacecraft dynamics**, the following files need to edited:

- **<eqShortName>CoreAlgorithm.m**

  If not already present in the function call, add the following input and output (as the dynamics contributions from all equipment are cumulated):

```
[ …, UnitsDyn] = <eqShortName>CoreAlgorithm(…, UnitsDyn);
```

  At any arbitrary point in the core model then assign force, torque and/or angular momentum contribution from the unit (for not assigned fields, the contribution is zero):

```
UnitsDyn.frc_B = UnitsDyn.frc_B + <any 3x1 value>;
UnitsDyn.trq_B = UnitsDyn.trq _B  + <any 3x1 value>;
UnitsDyn.angMom_B = UnitsDyn.angMom _B + <any 3x1 value>;
```

- **genericEquipmentModel.m**

  If not already present in the function call, add the following input and output (as the dynamics contributions from all equipment are cumulated):

```
[ …, UnitsDyn] = <eqShortName>CoreAlgorithm(…, UnitsDyn);
```

- **init<eqLongName>.m**

  Optionally, an initial value for the unit's dynamics contribution can be set in the subfunction `setInitialUnitsDynamics` (for not assigned fields, the initial contribution is zero).

4.7.2.2 Equipment Definition

Modifications in `defEquipments.m` (located in …`\data\def\properties`) are entirely related to the UIDs defined for an equipment. Apart from removing existing UIDs or adding new UIDs to be used in the equipment core algorithm, modifications to UIDs can be applied to:

- Change the output type of a UID
- Add/remove dynamic/kinematic states
- Add/remove actuation commands
- Modify the signal manipulations to be applied to a UID

Further details on the required parameters are provided in section 4.7.1.1.

4.7.2.3 Equipment Parameterization

This section describes all parameters which are present and can be modified in the parameterization file `param<EqShortName>.m` of an <Equipment>. The structure of the subsections represents the structure present in the parameter files. All parameter files for the specific equipment are located in the project or GAFE simulator subfolders …`\data\param\SEM\`.

A) Default values

All parameters in this section are considered valid for all units of an equipment. Thus, there is no need to repeat common parameters (e.g. sample time or noise properties) if more than one unit of the same equipment shall be used in a simulation. If individual parameter values need to be set for each unit (e.g. orientation in body frame), please refer to section **B)**.

A1) Nominal behaviour

Parameters defined in this section are entirely related to the "ideal performance model" of an equipment.

A1i) Common parameters

All parameters in this section are mandatory as being part of the generic equipment model, i.e. they cannot be removed or renamed; only their values can be changed. The size/validity of these parameters is automatically checked during initialization.

DCM from G to nominal U frame [-].This parameter must be provided as 3x3 direction cosine matrix.

```
User<EqShortName>.Default.Config.att_dcm_UnG          = <value>;
```

Nominal position of the unit in G frame [m]. This parameter must be provided as 3x1 vector.

```
User<EqShortName>.Default.Config.posUn_G              = <value>;
```

State transition table [-]

- Table of size [numStateIds x numStateIds], see defEquipmentStateIds()
- Tables values are transition types, see defTransitionTypeIds()

  (0 = no transition, 1 = manual, 2 = automatic)
- Columns: "from state"
- Row: "to state"
- Default tables in 'defDefaultTransitionTables()

```
User<EqShortName>.Default.Config.stateTransitionTable =
defDefaultTransitionTables('state');
```

State transition delays [s]:

- Table size identical to stateTransitionTable
- Tables values are transition delays, i.e.

  0 = no delay, >0 = delay in seconds
- Default tables in 'defDefaultTransitionTables()'

```
User<EqShortName>.Default.Config.transitionDelayTable =
defDefaultTransitionTables('delay');
```

Sampling time [s]:

Use -1 for 'continuous' sampling (i.e. sampled in each step of the fixed step simulation). If explicitly specified, the provided <value> must be an integer multiple of the fundamental simulation sampling time defined in `paramGeneral.m`.

```
User<EqShortName>.Default.Config.sampleTime           <value>;
```

Sampling time offset [s]:

If a (positive) non-zero <value> is provided, it must be an integer multiple of the fundamental simulation sampling time defined in `paramGeneral.m`.

```
User<EqShortName>.Default.Config.sampleOffset          <value>;
```

Sampling delay [s];

If a (positive) non-zero <value> is provided, it must be an integer multiple of the fundamental simulation sampling time defined in `paramGeneral.m`.

```
User<EqShortName>.Default.Config.sampleDelay           <value>;
```

A1ii) Equipment specific parameters

Parameters in this section are completely optional. No checks or conversions are applied unless explicitly set by the user in the respective initialization file `init<EqLongName>.m`, see section 4.7.1.4.

Parameters need to be mapped to the field 'Default.Config' field, e.g.

```
User<EqShortName>.Default.Config.<mySpecificParam>= <value>;
```

2) Nominal uncertainties

Parameters defined in this section are entirely related to the “nominal performance model” of an equipment.

A2i) Common parameters

All parameters in this section are mandatory as being part of the generic equipment model, i.e. they cannot be removed or renamed. The size/validity of these parameters is automatically checked during initialization.

% Two options for defining real unit frame:

a) 'random': Define real unit frame by uncertainties

b) 'explicit' Explicitly define real unit frame

```
User<EqShortName>.Default.Config.unitFrameOption =
‘random'/’explicit’;
```

Dependent on above selection, the parameters take effect:

a) case 'random'

Uncertainty in unit frame orientation [rad]

```
User<EqShortName>.Default.Config.tilt_UUn        = [value]
```

Uncertainty in unit frame position [m]

```
User<EqShortName>.Default.Config.posDeviationUUn = [value];
```

b) case 'explicit'

Position of real unit frame (U) in the geometry frame (G) [m]. This parameter must be provided as 3x1 vector.

```
User<EqShortName>.Default.Config.posU_G           = [value];
```

Orientation of real unit frame w.r.t. the geometry frame (G) [-].This parameter must be provided as 3x3 direct cosine matrix.

```
User<EqShortName>.Default.Config.att_dcm_UG       = [value];
```

A2ii) Signal manipulation parameters

The necessity of specifying parameters in this section depends on the nominal manipulation (to be) defined for this equipment (and its UIDs) in `defEquipments.m`

Parameters specified here are used to customize any kind of "nominal" signal manipulation in the respective equipment core algorithm (`<eqShortName>CoreAlgorithm.m`) to the assigned UID. To take effect, the respective UID itself needs to be defined in `defEquipments.m` for the <Equipment>.

For information about the required parameters of the various manipulations, please refer to `defSigMaTypes.m` and/or section 4.7.3.2.

Parameters need to be mapped to the field 'Default.SigMa', e.g.

```
User<EqShortName>.Default.SigMa.<UID>.<SigMaType>.<SigMaParam>(<perfId
x>) = <value>;
```

where:

- <UID> is the UID to which the parameters belong
- <SigMaType> is a string defining type of manipulation
- <SigMaParam> is a parameter required for the manipulation
- <perfIdx> (optional) is the performance index in case different performance modes are modelled for the equipment (see `defEquipmentQualityIds.m`). Indexing is only required in that case.

The size/validity of the parameter <value> is automatically checked during initialization. Missing parameters are automatically indicated.

If a signal manipulation is related to noise and multiple "noises of the same type" are required to combine the noise on a single UID (e.g. two Gauss-Markov processes to be summed up), the parameters can be mapped using an additional noise index <noiseIdx>:

```
User<EqShortName>.Default.SigMa.<UID>.<SigMaType>(<noiseIdx>)).<SigMaP
aram>(<perfIdx>) = <value>;
```

If an output signal related to a (dynamic/kinematic) state shall be limited this is defined explicitly via the InitState property in section 4) iv), not via the "Saturation" manipulation type as for "usual" UIDs. This is necessary to directly limit the integrator state properly rather than limiting its output value only.

A3) Fault behaviour

Parameters defined in this section are entirely related to the “fault performance model” of an equipment.

A3i) Unit fault properties

The modification of unit state transitions due to a fault is realized by providing a matrix of size (numMod x 4) [-]. Two types of modifications are possible which are defined row-wise; multiple changes are possible by defining multiple rows:

- a) (case = 1): change the transition type from one mode to another (`modValue` determines new transition type (no/manual/automatic = 0/1/2), see `defTransitionTypeIds()`;

```
[fromMode, toMode, case, modValue(=> 0,1,2)]
```

- b) (case = 2): change the delay of a (manual or automatic) transition

```
[fromMode, toMode, case, modValue(=> any value >=0 in [s])]
```

The operational states of a unit (`fromMode`, `toMode`) as defined in `defEquipmentStateIds()` are

- 1: unpowered
- 2: standby
- 3: initialization
- 4: operational
- 5: test mode (not for general use)

The necessity of specifying parameters in this section (apart from the generic transition fault depends on the presence of further transition faults (to be) defined for this equipment in `defEquipmentFaults.m`.

Parameters need to be mapped to the field:

```
User<EqShortName>.Default.Fault.<FaultName>.unitFault.Transition.param
Set = <values>
```

where

- <FaultName> is the name of a fault supporting transition faults for this equipment as defined in `defPhysicalFault.m`
- <values> is the numMod x 4 matrix as explained above

The size/validity of the parameter <values> is automatically checked during the initialization.

Examples:

Allow no transition from initialization to standby mode for a star tracker via the fault “STBERROR” (Note that the naming of the faults is generally arbitrary, but must be consistent with its definition in `defEquipmentFaults.m`):

```
UserStr.Default.Fault.STBERROR.unitFault.Transition.paramSet =
[2,3,1,0]
```

Set the automatic transition delay of a GNSS receiver from initialization to operational state to 5s via the fault “INITDELAY” :

```
UserGnsr.Default.Fault.INITDELAY.unitFault.Transition.paramSet =
[3,4,2,5]
```

A3ii) Signal manipulation fault parameters

The necessity of specifying parameters in this section depends on the presence of fault types (to be) defined for this equipment (and its UIDs) in `defEquipmentFaults.m`. For a list of all elementary fault types, please refer to `defElementaryFaults.m` or section 4.9.1.1.

Parameters specified here are used to customize any kind of "fault-case" signal manipulation in the respective equipment core algorithm (`<eqShortName>CoreAlgorithm.m`) to the assigned UID. To take effect, a fault affecting the respective UID needs to be defined in `defEquipmentFaults.m` for any fault of the <Equipment>.

For required parameters of the various manipulations, please refer to `defSigMaTypes.m` or section 4.7.3.2.

Parameters need to be mapped to the field `'Default.Fault'`, e.g.

```
User<EqShortName>.Default.Fault.<FaultName>.<UID>.<SigMaType>.<SigMaPa
ram> = <value>;
```

where

- <FaultName> is the name of a fault supporting the signal manipulation of a UID for this equipment
- <UID> is the UID to which the parameters belong
- <SigMaType> is a string defining type of manipulation
- <SigMaParam> is a parameter required for the manipulation

The size/validity of the parameter <value> is automatically checked during the initialization (and missing parameters are also automatically indicated in the command window output).

A4) Initial values

All parameters in this section are required to set the initial outputs and internal states of the Equipment module at simulation start (depending on the implementation in the core and the

presence of freeze faults, these values may or may not take effect, but must be defined for consistency).

A4i) HK outputs

Housekeeping outputs represent equipment outputs which are available to the AocsAlgo and the FdirOps module.

The necessity of specifying parameters in this section depends on the presence of UIDs which are (to be) defined to be associated with an HK output via the ".output" property in `defEquipments.m` for <MyEquipment>.

Parameters need to be mapped to the field:

```
User<EqShortName>.Default.InitHk.<UID> = <value>
```

where

- <UID> is the UID to which the HK output belongs
- <value> is the respective initial value

The size/validity of the parameter <value> is automatically checked during the initialization (and missing parameters are also automatically indicated.

A4ii) Status outputs (auxiliary)

Status outputs represent equipment outputs which have only auxiliary use for post-processing and analysis of a scenario. They are only logged, but not further made available to any other module.

The necessity of specifying parameters in this section depends on the presence of UIDs which are (to be) defined to be associated with a status output via the ".output" property in `defEquipments.m` for <MyEquipment>.

Parameters need to be mapped to the field:

```
User<EqShortName>.Default.InitHk.<Uid> = <value>
```

where

- <UID> is the UID to which the status output belongs
- <value> is the respective initial value

The size/validity of the parameter <value> is automatically checked during the initialization (and missing parameters are also automatically indicated.

A4iii) Operational and transition states

Apart from the operational states of the unit itself at startup, related timers can be used to simulate already initialised state transitions. The following parameters need to be present:

Current operational state [-] (see `defEquipmentStateIds` for a list possible states), e.g.

```
User<EqShortName>.Default.InitOpState.state = defEquipmentState.OFF;
```

Time passed since current state was reached [s], e.g.

```
User<EqShortName>.Default.InitOpState.time = 0;
```

Time passed since manual transition was initiated [s] (use NaN if no manual transition is initially triggered), e.g.:

```
User<EqShortName>.Default.InitOpState.manualTransitionTime = NaN;
```

Target operational state for the manual transition

- only has effect, if timer is not NaN
- only has effect if manual transition is allowed for transition from current to target state in provided state tables (see section **A1i)**).

```
User<EqShortName>.Default.InitOpState.manualTransitionTargetState =
NaN;
```

A4iv) Kinematic states

State outputs represent equipment outputs which any kind of kinematic/dynamic state which is fed to the "outside" integrator in the equipment module.

The necessity of specifying parameters in this section depends on the presence of UIDs which are (to be) defined to be associated with a state output via the ".output" property in `defEquipments.m` for <MyEquipment>.

Parameters need to be mapped to the following fields:

- User<EqShortName>.Default.InitState.Value.<UID> = <value>
- User<EqShortName>-Default.InitState.UpperLimit.<UID> = <value>
- User<EqShortName>.Default.InitState.LowerLimit.<UID> = <value>

where

- <UID> is the UID to which the state output belongs
- <value>is the respective initial value

The size/validity of the parameter <value> is automatically checked during the initialization. (and missing parameters are also automatically indicated).

B) Unit-specific values

This section can be empty.

All default parameters mentioned above can also be defined individually for each unit. In this case, the individual setting overwrites the default parameter. If nothing is specified in this section, only default above values are used.

Example:

Setting only `User<EqShortName>.Unit(2).Config.q_UnG = [0 1 0 0];` defines a specific orientation replacing `User<EqShortName>.Default.Config.q_UnG` for the second unit. All other units (and all other settings of unit 2) use the default values.

To explicitly define a seed for a specific noise quantity use the following form (exemplary for unit 2):

`User<EqShortName>.Unit(2).Config.Seed.<noiseType>(<noiseIdx>).<UID>(<perfIdx>) = <seedVector>;`

where <noiseType> and <UID> must match the fieldnames the respective noise model and variable and the <seedVector> must be of matching size (i.e. the size of the UID or twice its size in case of noiseType 'IntRandWalk'). The array indices <noiseIdx> and <perfIdx> do not have to be provided if only one performance index or noise model of a type is present for a UID.

4.7.2.4 Equipment Core Model

4.7.2.4.1 Model Core Algorithm

The core algorithm of each specific equipment model is based on a common skeleton as shown in the “template” below. The comments in red show where and how user manipulations can be applied. A further description of “replacement tags” can be found below the template.

```
The function syntax is mainly defined by the equipment properties in
defEquipments.m (see below this template for all dependencies).
function <%functionSyntax%>

This is the minimum set of persistent variables. If any further
definitions are required, they shall be declared as persistent here
and initialized in the first function call.
persistent isFirstCall DefStates DefStatus DefSubStates
if isempty(isFirstCall)
   DefStates = defEquipmentStateIds();
   DefStatus = defEquipmentStatusIds();
   DefSubStates = defEquipmentSubStateIds();
   isFirstCall   = false;
end

Status and housekeeping outputs defined in defEquipments.m are always
initialized using the value from the previous call; explicitly listed
additional HK outputs may not be removed (part of the GEM); manually
added Hk or Status outputs need to be added to these structures, see
description of the replacement token <%StatusOutputInit.UID%> and
<%HkOutputInit.UID%>

% Initialize output struct
StatusOutput = struct(...
   'opState', OpState.state, ...
   'opSubState', DefSubStates.NONE,...
   <%StatusOutputInit.UID%> );

HkOutput = struct( ...
   'isPwr', HkMem.isPwr.tab(1), ...
```

```
   'isCom', HkMem.isCom.tab(1), ...
   'isValidCnt', HkMem.isValidCnt.tab(1), ...
    <%HkOutputInit.UID%> );

Abbreviation of parameters mandatory for sample output determination;
may not be removed
% Sampling parameters
timeElapsed = Env.Time.timeElapsed;
sampleTime = UnitConfig.sampleTime;
sampleOffset = UnitConfig.sampleOffset;
sampleDelay = UnitConfig.sampleDelay;
simSampleTime = UnitConfig.simSampleTime;


Parameters mandatory as part of the GEM, may not be removed
% Current power/com status
isPwr = UnitConfig.StateTables.stateToStatusTable(DefStatus.PWR,
OpState.state);
isCom = UnitConfig.StateTables.stateToStatusTable(DefStatus.COM,
OpState.state);

Initialization of all UIDs
<%InitialiseSignal%>

Core model if unit is in operational state
% Compute output if operational
%=========================================================================
if ( OpState.state >= DefStates.OP)


This section can be arbitrarily edited based on all inputs, unit
configuration parameters and signal manipulations defined for the unit
to model the unit in operational state.


Update of all housekeeping memory based on the sampling properties
defined for the unit. the explicitly listed additional HK outputs may
not be removed (part of the GEM), manually added Hk outputs need to be
added to this function call, see description of the replacement token
<%HkOutputUpdate.UID%>

% Update HK memory
   HkMemUpdate = hkMemoryInput(...
      HkMem,timeElapsed,sampleTime,sampleOffset,sampleDelay,...
      simSampleTime,...
      'isPwr', isPwr, ...
      'isCom', isCom, ...
      'isValidCnt', timeElapsed, ...
       <%HkOutputUpdate.UID%> );

else
Core model if unit is in non-operational state
% Compute output if non-operational
%=========================================================================
```

```
This section can be arbitrarily edited based on all inputs, unit
configuration parameters and signal manipulations defined for the unit
to model the unit in non-operational state.


Update of all housekeeping memory based on the sampling properties
defined for the unit. The explicitly listed additional HK outputs may
not be removed (part of the GEM), manually added Hk outputs need to be
added to this function call, see description of the replacement token
<%HkOutputUpdate.UID%>
   % Update HK memory
   HkMemUpdate =hkMemoryInput(...
      HkMem,timeElapsed,sampleTime,sampleOffset,sampleDelay,...
      simSampleTime, ...
      'isPwr', isPwr, ...
      'isCom', isCom, ...
      'isValidCnt', HkOutput.isValidCnt , ...
      <%HkOutputInvalid.UID%> );
end

Update of all housekeeping outputs based on the sampling properties
defined for the unit.
% HK Output
[HkOutput,HkMemUpdate] = hkMemoryOutput(...
      HkMemUpdate,timeElapsed,sampleTime,sampleOffset,simSampleTime);

Assign all status outputs
Manually added Status outputs need to be added to this function call,
see description of the replacement token <%StatusOutput%>
<%StatusOutput%>

Assign the state derivative vector to be fed to the external
integrator (if states are defined for the unit)
<%StateDxOutput%>

Add the contribution of the unit to the overall contribution of all
equipments to S/C dynamics (force , torque, angular momentum)
<%UnitsDynamics%>

end
```

<%functionSyntax%>

This is a placeholder for the function syntax. The inputs and outputs are fully determined by the UID definition in `defEquipments.m`. The complete syntax for the function call is:

```
[HkOutput,HkMemUpdate,StatusOutput,stateDxVec,UnitNoiseData,UnitsDyn]

 = <eqShortName>CoreAlgorithm(...

 HkMem,StatusMem,stateVec,noiseVec,UnitNoiseData,UnitsDyn,unitCmd, ...
 Env,OpState,UnitConfig,SigMaParams)
```

Inputs:

- HkMem
  - The previous housekeeping data memory. Its size is automatically determined during initialization based on the relations between unit sampling delay and simulation sampling time.
  - **Mandatory** input
- StatusMem
  - The previous (auxiliary) status outputs. The structure is derived from all UIDs assigned to status and state outputs in `defEquipments.m` (plus additional generic status outputs from the GEM).
  - **Mandatory** input
- stateVec
  - A vector of unit states from the external integrator in the GEM. Its size depends on the size of the state vector output assigned in the equipment definition
  - **Dependent** input: only required if any of the UIDs is assigned to a state output in `defEquipments.m`
- noiseVec
  - A vector of standard normal noise samples (from the external noise generator in the GEM) to be used for signal manipulation. Its size is automatically determined based on the noise manipulations defined for all UIDs in the nominal and fault case.
  - **Dependent** input: only required if a noise manipulation type has been assigned to any of the equipment UIDs in either the nominal case (`defEquipments.m`) or in the fault case (`defEquipmentFaults.m`)
- UnitNoiseData
  - A structure containing the noise filter states and noise vector indices for all UIDs which are related to a noise manipulation. The structure content is automatically determined based on the noise manipulations defined for all IDs in the nominal and fault case.
  - **Dependent** input: only required if a noise manipulation type has been assigned to any of the equipment UIDs in either the nominal case (`defEquipments.m`) or in the fault case (`defEquipmentFaults.m`)
- UnitsDyn
  - A structure containing the total contribution to the S/C dynamics collected from all equipment units (without the current equipment unit); the structure has fixed fields for force (frc_B), torque (trq_B) and angular momentum (angMom_B) of size 3x1 each.
  - **Dependent** input: only if an actuation command flag has been assigned to an equipment UID in `defEquipments.m`.
- unitCmd

    - The unit actuation command vector from the AOCS algorithms. Its size corresponds to the size of the UIDs to which the actuation command flag has been assigned in `defEquipments.m`.
    - **Dependent** input: only if an actuation command flag has been assigned to an equipment UID in `defEquipments.m`.
- Env
    - A large structure which contains all possible information from the environment module (S/C states, time, ephemeris, physical environment, etc.)
    - **Mandatory** input (assuming that all sensor/actuator require at least basic information about their environment)
- OpState
    - A structure of fixed size containing the unit operational state and information about ongoing state transitions.
    - **Mandatory** input
- UnitConfig
    - All unit configuration properties derived from parameter defined in `param<EquipmentLongName>.m`
    - **Mandatory** input
- SigMaParams
    - The set of applicable signal manipulation parameters for all UIDs. The selection of applicable parameters (either a neutral parameterization, nominal or fault manipulations) is carried out in the GEM.
    - **Mandatory** input

Outputs:

- HkOutput
    - The actual unit housekeeping data output for the current simulation step. The structure is derived from all UIDs assigned to HK outputs in `defEquipments.m` (plus additional generic outputs from the GEM).
    - **Mandatory** output
- HkMemUpdate
    - The updated housekeeping data memory. The structure matches the respective structure of the HkMem input.
    - **Mandatory** output
- StatusOutput
    - The updated (auxiliary) status outputs. The structure matches the respective structure of the StatusMem input.

    - **Mandatory** output
- stateDxVec
    - The unit state derivative vector which is fed to the external integrator of the GEM. Its size matches the size of the corresponding state vector input (stateVec).
    - **Dependent** output: same conditions as for state vector input apply
- UnitNoiseData

  Noise filter states and vector indices for all UIDs [-] <struct>
    - A structure containing the updated noise filter states and (fixed) noise vector indices for all UIDs which are related to a noise manipulation. The structure matches the size of the equivalent input.
    - **Dependent** output: same conditions as for the respective input apply
- UnitsDyn

  Total contribution to S/C dynamics including the current unit [-] <struct>
    - The total contribution to the S/C dynamics collected from all equipment units including the current equipment unit. The structure matches the size of the equivalent input.
    - **Dependent** output: same conditions as for the respective input apply

These “generic” I/O arguments can however be extended, e.g. in case a further memory variable is required (e.g. an internal timer). In this case, the memory variable has to be assigned both as input and output argument. This is necessary as the memory “management” has to be carried out on GEM level (due to multiple possible instances/units of one equipment).

<%StatusOutput.UID%>

This placeholder represents the assignment of all status output UIDs (defined in `defEquipments.m`) to zeros of proper size.

```
'<statusUidName>', zeros(size(<statusUidName>) ...
```

Example (reaction wheel core):

```
   'actTrqCmd',0,...
   'actTrq', 0,...
   'rwFrctTrq',0,...
   'rwTrq_B', zeros(3,1),...
   'rwRate',0, ...
```

<%HkOutputInit.UID%>

This placeholder represents the assignment of all housekeeping output UIDs (defined in `defEquipments.m`) to the values from the previous function call (first column in memory tab).

```
'<hkUidName>', HkMem.<hkUidName>.tab(:,1) ...
```

Example (reaction wheel core):

```
   'rwRateMeas', HkMem.<hkUidName>.tab(:,1), ...
```

<%InitialiseSignal%>

This placeholder represents the local initialization of all UIDs with zeros of proper size, independent of the output type. This zero-value assignment is just a dummy which needs to be replaced by the actual model core functions

<%HkOutputUpdate.UID%>

This placeholder represents the update of the memory for housekeeping UIDs (defined in `defEquipments.m`) in case the unit is in operational state.

```
'<hkUidName>', <hkUidName> ...
```

<%HkOutputInvalid.UID%>

This placeholder represents the update of the memory for housekeeping UIDs (defined in `defEquipments.m`) in case the unit is in non-operational state.

```
'<hkUidName>', NaN(size(HkOutput.<hkUidName>))...
```

<%StatusOutput%>

This placeholder represents assignment of all status output UIDs (defined in `defEquipments.m`) to the status output structure.

```
StatusOutput.<statusUidName> = <statusUidName> ;
```

<%StateDxOutput%>

This placeholder represents assignment of any variable computed in the model core algorithm to the state derivative vector. (defined in `defEquipments.m`) to the status output structure.

```
stateDxVec = zeros(size(stateVec)) % Replace by actual value!;
```

Note: As the actual assignment cannot be automatically generated a priori (when `createEquipment.m` was used to generate the equipment files), the “zeros”-placeholder need to manually replaced during the core model development.

<%UnitsDynamics%>

This placeholder represents assignment of any variables computed in the model core algorithm to the overall unit force/torque/angular momentum contributions to the spacecraft.

```
% Replace zeros by actual values where applicable!
UnitsDyn.frc_B = UnitsDyn. frc _B  + zeros(3,1);
UnitsDyn.trq_B = UnitsDyn. trq _B  + zeros(3,1);
UnitsDyn.angMom_B = UnitsDyn.angMom_B  + zeros(3,1) ;
```

Note: As the actual assignments cannot be automatically generated a priori (when `createEquipment.m` was used to generate the equipment files), the “zeros”-placeholders need to manually replaced during the core model development.

4.7.2.4.2 Calling the SEM Core from the GEM

When modifying an existing [Equipment], the function call in the GEM only has to be adapted if the modification introduces new inputs or outputs. The function call of the SEM is encapsulated in a switch-case expression in the GEM. Its syntax obviously corresponds to one of the specific core algorithm file (described in the previous section). This content this switch-case expression is shown below for “complete” I/O relations:

```
case 'Example'

[Hk.(nameEq{idxEq})(idxUnit), ...
   HkMem.(nameEq{idxEq})(idxUnit),...
   Status.(nameEq{idxEq})(idxUnit),...
   unitsDxVec(Config.(nameEq{idxEq})(idxUnit).StateIndex.global),…
   NoiseData.(nameEq{idxEq})(idxUnit)] ...
   = exampleCoreAlgorithm(...
   HkMem.(nameEq{idxEq})(idxUnit),...
   Status.(nameEq{idxEq})(idxUnit),...
   unitsStateVec(Config.(nameEq{idxEq})(idxUnit).StateIndex.global),…
   noiseVec(NoiseData.(nameEq{idxEq})(idxUnit).globalIndex),...
   NoiseData.(nameEq{idxEq})(idxUnit),...
   Env,...
   CurrentOpState.(nameEq{idxEq})(idxUnit),...
   Config.(nameEq{idxEq})(idxUnit),...
   Manipulation.(nameEq{idxEq})(idxUnit).Parameters);
```

In case a further memory variable (such as an internal timer) is required for the model, this memory variable has to be assigned both as input and output argument and made persistent on GEM level:

```
case 'Example'

% Initialize persistent variable for all units
if isempty(isFirstEqCall)
   myPersistentVariable = zeros(thisEqNumUnits,1);
   isFirstEqCall = false;
end

[Hk.(nameEq{idxEq})(idxUnit), ...
   HkMem.(nameEq{idxEq})(idxUnit),...
   Status.(nameEq{idxEq})(idxUnit),...
   unitsDxVec(Config.(nameEq{idxEq})(idxUnit).StateIndex.global),…
   NoiseData.(nameEq{idxEq})(idxUnit) ...
   myPersistentVariable(idxUnit)] ...
   = exampleCoreAlgorithm(...
   HkMem.(nameEq{idxEq})(idxUnit),...
   Status.(nameEq{idxEq})(idxUnit),...
```

```
    unitsStateVec(Config.(nameEq{idxEq})(idxUnit).StateIndex.global),…
    noiseVec(NoiseData.(nameEq{idxEq})(idxUnit).globalIndex),...
    NoiseData.(nameEq{idxEq})(idxUnit),...
    Env,...
    CurrentOpState.(nameEq{idxEq})(idxUnit),...
    Config.(nameEq{idxEq})(idxUnit),...
    Manipulation.(nameEq{idxEq})(idxUnit).Parameters, ...
    myPersistentVariable(idxUnit));
```

4.7.2.5 Equipment Faults

Changing the parameters of existing faults is realized by simply adapting the desired values in section **A3ii)** in `param<EqLongName>.m` of the <Equipment>. Introducing entirely new faults to the fault behaviour of the equipment is described in section 4.9.1. In case these faults require user parameters, they need to added to section **A3ii)** as well. Any missing parameter is detected during the initialization and a respective error message is displayed.

4.7.2.6 Equipment Plot Data Configuration

Each equipment has a dedicated configuration file for related plots. These files are located in folder `<gaferoot>\data\plotCfg` and named `<EqLongName>_cfg.m`. In case the plot configuration shall only be adapted for a project, the respective file need to be copied simply to the equivalent project path (i.e. `<projectroot>\data\plotCfg`). The configuration files for user-created equipments in a project are automatically located in the project as well.

If any Status or Housekeeping outputs have been manually added to an existing equipment, the plot configuration for these outputs (e.g. title, yLabels, etc.) can be configured in the respective `_cfg` file.

See also section 4.4.6.1 for more information about plotting and plot configuration.

4.7.3 Signal Manipulation

4.7.3.1 Concept

- All signal manipulations provide a *neutral parameterization*, which leads to: sigOut = sigIn (i.e. no modification between input and output). Thus, the *neutral parameterization* can be used to set a sensor/actuator to “Ideal Performance”.
- Calls for signal manipulations are hard coded at specific locations inside the core models (see exemplary code in Table 4-2)
- Individual calls of signal manipulations are not differentiated a-priori into “belonging to nominal performance model" and “belonging to fault performance model". Differentiation is only done by parameterization (see next bullet)
- There is one dedicated parameterization for the nominal performance case (no fault present) and for every specific fault case (e.g. fault 2 present), see Table 4-3.
- If no fault is present, the nominal parameterization is used. For manipulations whose effects are not desired in the nominal case (e.g. output freeze) the *neutral parameterization* (see section 4.7.3.2) is used.

- If a single fault is present, all parameters of manipulations defined for this fault become applicable and overwrite the nominal ones. If specific parameters are not specified for a fault (indicated by a "-" in Table 4-4), the nominal parameters stay applicable.
- If more than one fault is present, the injection times (or sequence in case of injection at the same time) of the faults determine the parameterization finally used in the core model (see example in Table 4-4).

Implementation:

- Every call to a signal manipulation shall be identified by a unique ID (UID, unique per unit). This UID shall be reflected in parameterization and in naming of states.

  Example for reaction wheel UID definition and nominal manipulations (`defEquipments.m`); only parameters relevant for this example shown:

```
%--------------------------------------------------------------------
Def.Rw.Uid(1).name            = 'rwRateMeas';
Def.Rw.Uid(1).nomSigMa        = {...
   DefSigMa.GAUSSMK.str ...
   DefSigMa.QUANT.str};
%--------------------------------------------------------------------
Def.Rw.Uid(2).name            = 'actTrqCmd';
Def.Rw.Uid(2).nomSigMa        = {...
   DefSigMa.SATURATION.str};
%--------------------------------------------------------------------
Def.Rw.Uid(4).name            = 'coulombTorque';
Def.Rw.Uid(4).nomSigMa        = {};
%--------------------------------------------------------------------
Def.Rw.Uid(5).name            = 'viscFrctGain';
Def.Rw.Uid(5).nomSigMa        = {};
%--------------------------------------------------------------------
Def.Rw.Uid(6).name            = 'actTrq';
Def.Rw.Uid(6).nomSigMa        = {...
   DefSigMa.BLWN.str...
   DefSigMa.SCALE.str};
```

  Example for reaction wheel fault definition (`defEquipmentFaults.m`); only parameters relevant for this example shown:

```
%--------------------------------------------------------------------
% Fault 1: Tachometer bias:
Fault.Rw.TACHOBIAS.rwRateMeas      = [Id.BIAS];

% Fault 2: Friction & temperature
Fault.Rw.FRICTEMPINC.coulombTorque = [Id.BIAS];
Fault.Rw.FRICTEMPINC.viscFrctGain = [Id.BIAS];
Fault.Rw.FRICTEMPINC.rwRateMeas = [Id.BIAS];

% Fault 3: Wrong torque command
Fault.Rw.WRONGTRQ.actTrqCmd        = [Id.SCALE,Id.BIAS];
```

  Example for reaction wheel parameterization (`paramReactionWheel.m`); note that only "real" user values are required, flags values (isEnabled) are automatically generated:

```
% Nominal parameters
UserRw.Default.SigMa.rwRateMeas.GaussMk.std            = ...
UserRw.Default.SigMa.rwRateMeas.GaussMk.tCorr          = ...
UserRw.Default.SigMa.rwRateMeas.Quantization.step      = ...
UserRw.Default.SigMa.actTrq.Blwn.std                   = ...
UserRw.Default.SigMa.actTrq.SymScaleErr.value          = ...
UserRw.Default.SigMa.actTrqCmd.Saturation.maxValue     = ...
UserRw.Default.SigMa.actTrqCmd.Saturation.minValue     = ...

% Fault 1 parameters (Tachometer bias)
UserRw.Default.Fault.TACHOBIAS.rwRateMeas.Bias.value      = ...

% Fault 2 parameters (Friction & temperature increase)
UserRw.Default.Fault.FRICTEMPINC.coulombTorque.Bias.value = ...
UserRw.Default.Fault.FRICTEMPINC.viscFrctGain.Bias.value  = ...
UserRw.Default.Fault.FRICTEMPINC.rwRateMeas.Bias.value  = ...

% Fault 3 parameters (Wrong torque command)
UserRw.Default.Fault.WRONGTRQ.actTrqCmd.Bias.value        = ...
UserRw.Default.Fault.WRONGTRQ.actTrqCmd.SymScaleErr.value = ...
```

- The GEM keeps track of which faults are present and when they were injected based on the inputs from the fault injection module.

  Example:

  Fault1: present, injection time: 200s

  Fault2: present, injection time: 100s

  Fault3: not present

Table 4-2: Exemplary (pseudo-)code including hard coded signal manipulation function calls for reaction wheel .

```
function [outputs] = genericEquipmentModel(inputs, params)

   % Management of parameter selection here: determine currently applicable
     parameters; no need of user modifications

   ApplicableParams = fct(presentFaults)

   function [out, ...] = rwCoreAlgorithm(in, ApplicableParams...)
   someLinesOfCode ...
   % UID: rwRateMeas
   [rwRateMeasOut,  ...]= manipulateSignal(...
       rwRateMeasIn,ApplicableParams,'rwRateMeas',...);

   someLinesOfCode ...

   % UID: actTrqCmd
   [actTrqCmdOut,  ...]= manipulateSignal(...
       actTrqCmdIn,ApplicableParams,'actTrqCmd',...);

   someLinesOfCode ...

   % UID: actTrq
   [actTrqOut,  ...]= manipulateSignal(...
       actTrqIn,ApplicableParams,'actTrq',...);
```

```
   someLinesOfCode ...

   % UID: coulombTorque
   [coulombTorqueOut,  ...]= manipulateSignal(...
       coulombTorqueIn,ApplicableParams,'coulombTorque',...);

   % UID: viscFrtGain
   [viscFrctGainOut,  ...]= manipulateSignal(...
       viscFrctGainIn,ApplicableParams,'viscFrctGain',...);
   someLineOfCode ...

   end;
end;
```

Table 4-3: Parameterization of individual sigma-functions for nominal & fault cases.

Function Call UID	Parameter(s)	Nominal	Fault 1	Fault 2	Fault 3	…
rwRateMeas	Quantization.step	1.0	-**	-	-	…
rwRateMeas	GaussMk.std GaussMk.tCorr	1.0 1.1	-	- -	-	…
rwRateMeas	Bias.value	NP*	0.05	0.07	-	…
actTrqCmd	Saturation.minValue Saturation.maxValue	-2000 2000	-	0.03 0.4	-	…
actTrqCmd	Bias.value	NP*	-	-	0.02	…
actTrqCmd	Scale.value	NP*	-	-	0.02	…
actTrq	Blwn.std	0.001	-	-	-	…
actTrq	Scale.value	0.01	-	-	-	…
viscFrctGain	Bias.value	NP*	-	0.03	-	…
coulombTorque	Bias.value	NP*	-	0.04	-	…
…	…	…	…	…	…	…

*) “NP”: neutral parameterization.

**) “-“: not specified

Table 4-4: Applicable parameters (orange background) for sigma-functions inside SEM at t = 300s in case faults 1 & 2 are present and the fault injection times are $t_{inj,fault3}$ = 100s, $t_{inj,fault1}$ = 200s.

Function Call UID	Parameter(s)	Nominal	Fault 1	Fault 2	Fault 3	…

rwRateMeas	Quantization.step	1.0	-**	-	-	…
rwRateMeas	GaussMk.std GaussMk.tCorr	1.0 1.1	-	- -	-	…
rwRateMeas	Bias.value	NP*	0.05	0.07	-	…
actTrqCmd	Saturation.minValue Saturation.maxValue	-2000 2000	-	0.03 0.4	-	…
actTrqCmd	Bias.value	NP*	-	-	0.02	…
actTrqCmd	Scale.value	NP*	-	-	0.02	…
actTrq	Blwn.std	0.001	-	-	-	…
actTrq	Scale.value	0.01	-	-	-	…
viscFrctGain	Bias.value	NP*	-	0.03	-	…
coulombTorque	Bias.value	NP*	-	0.04	-	…
…	…	…	…	…	…	…

4.7.3.2 Elementary Signal Manipulation Types

The elementary signal manipulation types which can be used for the “nominal performance model” are defined in the file `defSigMaTypes.m` which is located in the folder …`\data\def\properties\`. The list below describes all possible manipulations based on the <SigMaType> and its required <SigMaParams> (italic) together with their neutral parameterization:

- BIAS

  Applies an additive bias with a given *value* to the signal associated with a UID. The bias *value* must match the size defined for the UID.

  Neutral parameterization: *value* = zeros(size(UID))

- BLWN

  Adds a band-limited white noise of a given standard deviation to the signal associated with a UID. This *std* must match the size defined for the UID. No cut-off frequency is explicitly defined, thus the simulation sampling frequency implicitly limits the effective frequency band .

  Neutral parameterization: *std* = zeros(size(UID))

- GAUSSMK

  Adds a 1st order Gauss-Markov low-pass filtered noise to the signal associated with a UID. Its standard deviation (*std*) and correlation time (*tCorr)* must match the size defined for the UID.

  Neutral parameterization:

- *std* = zeros(size(UID))
- *tCorr* = ones (size(UID))

- RANDWALK

  Adds a random walk to the signal associated with a UID defined by a random walk coefficient (*randWalkCoef*) which must match the size of the UID.

  Neutral parameterization: *randWalkCoef* = zeros(size(UID))

- INTRANDWALK

  Adds a integrated random walk to the signal associated with a UID defined by an integrated random walk coefficient (intR*andWalkCoef*) which must match the size of the UID.

  Neutral parameterization: *intRandWalkCoef* = zeros(size(UID))

- LTINOISE

  Adds a noise with a spectrum “shaped” via (the transfer function magnitude of) an LTI model to the signal associated with a UID.

  The necessary state space matrices A,B,C,D must be constructed such that the output size of the LTI models matches the dimension specified for the UID.

  Neutral parameterization:

  - *noiseSsA* = zeros(1,1)
  - *noiseSsB* = zeros (1, 1)
  - *noiseSsC* = zeros (dim(UID), 1)
  - *noiseSsD* = zeros (dim(UID), 1);

  Note: in case no nominal LTI-noise manipulation is defined, but only a fault manipulation, the neutral parameterization is adapted such that it matches the size of the latter (and vice versa).

- SCALE

  Adds a symmetric scale error of the form (1+ *value*) * signal to the signal associated with a UID. The *value* of the scale error must match the size of the UID.

  Neutral parameterization: *value* = zeros(size(UID))

- SCALEA

  Adds a asymmetric scale error of the form (1+ sign(signal) * *value*) * signal to the signal associated with a UID. The *value* of the scale error must match the size of the UID.

  Neutral parameterization: *value* = zeros(size(UID))

- QUANT

  Applies a quantization of customizable step to the signal associated with a UID. The size of the quantization *step* must match the size of the UID. Optionally, also a quantization *method* (‘round’ or ‘floor’) can be explicitly specified.

Neutral parameterization:

- *value* = zeros(size(UID))
- *method* = ‘round’ (default)

- SATURATION

  Applies a limiter to the signal associated with a UID by providing upper and lower limits. The respective parameters *maxValue* and *minValue* must match the size of the UID.

  Neutral parameterization:

  - *minValue* = -Inf(size(UID))
  - *maxValue* = Inf(size(UID))

There are additional manipulations available for fault events which are described in section 4.9.1.1.

4.7.3.3 Guidelines and Background

4.7.3.3.1 Manipulation Sequence

The sequence in which the manipulations are applied to a single UID is fixed as follows:

maniSignal = (SCALE +SCALEA) * inputSignal

+ BIAS

+ (BLWN + GAUSSMK + RANDWALK + INTRANDWALK + LTINOISE)

The application of quantization and saturation depends on the specified “type” of the UID in defEquipments.m (vector, unit vector or quaternion).

For “vector” type signals, quantization and saturation are directly applied, i.e.

outputSignal = SATURATION(QUANT(maniSignal)

For “unit vector” and “quaternions”, the manipulated signal represents and angular error which first needs to be converted to the correct format (the size of the UID needs to be 3 in both cases and the inputSignal does not correspond to the nominalSignal, but is a zero signal of this size). Here the sequence is as follows:

outputSignal = QUANT(CONVANGERR(SATURATION(maniSignal)))

The conversion of the angular error (CONVANGERR) in the “unit vector” case is:

- Convert the angular error per axis into an error quaternion
- Rotate the nominalSignal using the computed quaternion

The conversion of the angular error (CONVANGERR) in the “quaternion” case is:

- Convert the angular error per axis into an error quaternion
- Multiply the nominalSignal with the error quaternion

4.7.3.3.2 User-defined Manipulation Sequence

Although the sequence above is fixed, basically any manipulation sequence can be realized by defining additional “auxiliary” UIDs.

Example:

Assume a UID is associated with a command input of a unit which is limited between certain bounds (e.g. torque command of a reaction wheel). The actually applied command to the unit is further subject to noise and bias.

Defining all manipulations for a single ID, the manipulation sequence would not only limit the command itself, but the entire corrupted signal. As a consequence even the noise would be strictly cut when exceeding the defined bounds.

If this is not desired, one UID for the commanded signal (saturation only) and one UID for the realized command (bias and noise) can be defined alternatively where the saturated input command is then input to the second manipulation.

4.7.3.3.3 Disabling Manipulations

If the signal manipulation for a certain UID shall be completely (or just partially, e.g. only the bias component) disabled for a simulation run, the respective parameters simply have to be set to the “neutral” parameterization in the parameter file of the equipment. There is no need to remove the manipulation from the equipment definition or to modify the core algorithm.

4.7.3.3.4 Implementation

The signal manipulation function (`manipulateSignal`) needs to be called in the core algorithm of the equipment separately for each UID. The syntax is:

```
[actualSignal, NoiseData] = manipulateSignal(...
nominalSignal, ...
SigMaParams, ...
previousSignal, ...
sampleTime, ...
noiseVec, ...
NoiseData, ...
signalName, ...
performanceIndex)
```

where:

- *nominalSignal* is the nominal input signal to be manipulated
- *SigMaParams* is the structure of applicable signal manipulation parameters
- *previousSignal* is the "memorized signal" from the previous function call (required for freeze faults)
- *sampleTime* is the sample time of the current equipment unit
- *noiseVec* is the vector of standard normal samples as input for the noise models defined for this UID
- *NoiseData* is the structure of noise model states and the local index in the noise vector for all noise models of the UID (and its updated version as output)
- *signalName* is the name of the UID itself (as string)
- *performanceIndex* is the (optional value) for the performance index as defined in defEquipmentQualityIds.m
- *actualSignal* is the manipulated input signal

4.8 Developing own AOCS Algorithms

In addition to the predefined AOCS algorithmic components, see chapter 5.2, the user can develop his own algorithms.

A set of functionality is offered, which simplifies this development.

Follow this workflow for developing the new Component “aocsAlgoComponent”:

- First create the Component at place, preferably you are in the “src/AocsAlgo” folder of your project using: createAocsAlgoComponent(<aocsAlgoComponentName>).
  - The programmatic interface of each component follows a common pattern: [Out, Status, States, Params] = <componentName>(getDefs, Params, <In1>, <In2>, …, States). This function will be called with and without inputs, so please take care, that this is working.
  - Read the hints in the created m-file before developing the algorithm inside
  - Develop the algorithm
  - Inside the created Component you will find a comment section with this workflow again, together with some useful expressions that you can copy & paste into the MATLAB command window to simplify usage.
- Generate the dsc-file using "generateAocsAlgoComponentDsc(<aocsAlgoComponentName>)"
  - A doc-folder will be created, the dsc-file will be placed in it and directory switch will be performed
  - Fill missing parts of dsc-file

    - Update the Component header by typing: s=loadDscFile('<aocsAlgoComponentName>') and "updateMFileHeader(s)"
- Now test your algorithm, preferably using a unit test in open loop testing the Component only
- Generate dsc-file for Interfaces using "generateAocsAlgoComponentIoDsc(<aocsAlgoComponentName>)"
    - Fill missing parts of dsc-file for interfaces
- Generate cfg-file using "generateAocsAlgoComponentIoCfg(<aocsAlgoComponentName>)"

Some rules and hints :

- All Controller should have identical output and deliver “0” when not used, see “rateDampCtrl”. The function “dynCmdDistribution” shall be used to distribute dynamics to the different actuators. Modify it for your usage
- Each Component returns an “isValid”, signalising if it’s outputs can be used in next Components
- The “Status” should be structured with items with the name prefix “is” using the predefined values of “defAocsAlgoIsValidIds”
- Sampling times can be assigned to each algorithmic component. This is very simple for the user by defining the following “Params” in his algorithm
    - `Params.sampleRate` [Cnts]
    - `Params.sampleOffset` [Cnts]
    - `Params.sampleNoHitHoldOutputs` [0/1], signalising what to do with the outputs when sampling not hit
    - `Params.sampleNoHitResetStates` [0/1], signalising what to do with the states when sampling not hit

  This is optional, if not used, the sampling rate is assumed to be 1. The “AOCS Algorithmic Component” does not need to do more in its algorithm, all is managed by the “AocsAlgo” module.

More Rules and hints are listed in the created Component M-File.

4.9 Defining and Applying Faults

This section covers the definition of faults and their application during a simulation run. Faults can affect equipment units, the AOCS algorithms and the S/C system module.

4.9.1 Equipment Faults

As for the nominal signal manipulation, a "physical" fault applied to an equipment unit is set up using a combination of elementary fault types. These fault types either affect a single UID defined for a unit or the operational state of an entire unit. The following subsection first

describes the elementary faults available and second how they these are used to defined a "physical" fault for an equipment.

4.9.1.1 Elementary Fault Types

The elementary fault types contain all options which are also related to the "nominal" signal manipulations described in section 4.7.3.2. Further, the following "fault only" types are available broken down into "subtypes":

Unit faults (applicable to an entire unit, not only to a single UID)

- NOPOW

  Applies a power failure to a unit by setting its power status in the state transition tables to "false".

  Neutral parameterization (i.e. parameter value where the manipulation has no effect): *isEnabled* = 0

- NOCOM

  Applies a communication failure to a unit by setting its communication status in the state transition tables to "false"

  Neutral parameterization: *isEnabled* = 0

- TRANS

  Applies a fault to the (operational) state transition tables of a unit using a parameter set (paramSet) which determines the entry and type of modification in the transition tables (see section 4.7.2.3, **A3i** for further information on the *paramSet* content).

  Neutral parameterization:

  - *isEnabled* = 0
  - *paramSet* = zeros(1,4)

Non-parametric faults (require no further user-parameters)

- NOVAL

  Overwrites the value of a UID by an invalid value using a NaN -vector of proper size.

  Neutral parameterization: *isEnabled* = 0

- FREEZE

  Freezes the value of a UID by holding the value stored from a previous function call.

  Neutral parameterization: *isEnabled* = 0

- INVSIGN

  Inverts the sign of a signal (element-wise to vector components) associated with a UID.

  Neutral parameterization: *isEnabled* = 0

Parametric faults (require further user-inputs in the parameter file of the equipment)

- FIXVAL

Overwrites the value of a UID by a fixed user-defined parameter *value*.

Neutral parameterization:

- *isEnabled* = 0
- *value* = 0

- PERMAX

  Permutes the axes of a vector output signal by providing the axis *sequence* as parameter (e.g. [3 2 1] permutes the x- and z- axis of a 3D vector signal).

  Neutral parameterization: *sequence* = [1:1:size(UID)]

- OUTLIER

  Introduces random outliers to a signal by scaling the nominal value of the signal with a user-defined *factor*. The probability of the outlier is further specified via the value of the *occurence* parameter between 0 (no occurence) and 1 (100% occurence).

  Neutral parameterization:

  - *factor* = 1
  - *occurence* = 0

4.9.1.2 Fault Definition

The fault definition for all equipment faults is based on the file `defEquipmentFaults.m` located in the GAFE simulator subfolder …`\data\def\properties\`.

To modify the fault definition for an existing equipment <Eq>, just provide or extend the entries in the Def structure in the form:

```
Fault.<EqShortName>.<FaultName>.<UID>   =  <IDs>
```

with:

- <FaultName>: An arbitrary identifier for the fault used to inject/eject the fault in paramFaultInjection.m (see section 4.9)
- <UID>: The identifier of the "signal" be manipulated by the fault in the equipment core **Exception**: "unitFault" can be used as <UID> to introduce a fault affecting the operational state of a unit (or their transitions)
- <IDs>: A single ID or vector of IDs to specify the applicable elementary fault types, see previous section)

Example: Temperature increase fault on star tracker which affects the measurement (noise increase) and quality index (bias):

```
Fault.Str.TEMP.satAttMeas_euler_UJ    = [Id.BLWN];
Fault.Str.TEMP.satAttQualIndex        = [Id.BIAS];
```

Example: Measurement degradation fault on accelerometer which affects the measurement acceleration (increased noise parameters of various noise models):

```
Fault.Acc.MEASDEGRAD.satNonGravAccMeas_U  =
[Id.BLWN,Id.GAUSSMK,Id.RANDWALK]
```

Example: Complete failure of an Earth sensor which affects its operational status and sets and the validity of the measured output:

```
Fault.Es.COMPLETE.unitFault           = [Id.NOPOW, Id.NOCOM];
Fault.Es.COMPLETE.satEarthDirMeas_U   = [Id.NOVAL];
```

4.9.2 Injecting/Ejecting Faults

The timeline of all faults which are to be injected or ejected during a simulation run is entirely defined using the function `paramFaultInjection.m`, i.e. there are no separate files for faults to be applied to the Equipment, AocsAlgo orSystem module.

Note that there is no “accumulation” of fault parameters, i.e. a fault injected twice does not double or increase its effect on a certain quantity.

For each fault event, a structure `UserFaultInjection(idx)` must be provided with following fields (idx denotes the consecutive numeric index of the events) :

- .module

  The ID of the module to which the fault applies. Possible options are:

  - 'Equipment'
  - 'System'
  - 'AocsAlgo'

- .time

  Simulation time at which the event occurs [s].

  Note:

  - In case the time defined for the fault event is smaller than the simulation time specified in `paramGeneral.m`, the event has no effect and a respective warning is displayed during initialization.
  - Multiple faults can be injected at a time to a given module and unique parameters in the simultaneous events all take effect, however, as described above, there is no accumulation of faults on a single parameter. This means, in case more than one of these simultaneous events affect the same quantity (e.g. one specific UID of an equipment unit), the fault with the higher index (idx) overwrites the common parameters from the “previous” fault event (i.e. the event with the smaller index).

    Example: Fault1 affects the standard deviation and bias on UID1, Fault2 effects the standard deviation and scale error on UID1. Both fault are injected at the same time. If Fault 2 has the larger index, standard deviation and scale error of Fault2 as well as the bias of Fault1 are acting on UID1.

- .presence

  Injection or ejection of fault at given time. Possible values are:

  - 0: Ejection
  - 1: Injection

- .persistency

  Persistency of a fault. Non-persistent faults are automatically ejected after restart or reboot of the system or a unit. Persistent fault remain present until they are explicitly ejected via the 'FaultInjection' module. The following values are possible:

  - 0: Non-Persistent
  - 1: Persistent

- .type

  The fault type to be applied defined needs to be defined as a string. This string is dependent on the module to which the fault applies:

  - for 'Equipment' faults, see defEquipmentFaults()
  - for 'System' faults, see defSystemFaultIds()
  - for 'AocsAlgo' faults, see defAocsAlgoFaultIds()

- .component

  The string for the component is again dependent on the module to which the fault applies:

  - for 'Equipment' faults, the equipment short name see defEquipments()
  - for 'System' faults, see defSystemComponentIds()
  - for 'AocsAlgo' faults, see defAocsAlgoComponentIDs()

- .unit

  This parameter only has an effect on 'Equipment' module faults). It defines the unit number of the equipment to which the event applies.

Example fault event: Inject non-persistent freeze on the measurement HK output of the 2nd magnetometer at t=1000s:

```
idx = idx + 1;
UserFaultInjection(idx).module      = 'Equipment';
UserFaultInjection(idx).component   = 'Mag';
UserFaultInjection(idx).unit        = 2;
UserFaultInjection(idx).time        = 1000;
UserFaultInjection(idx).presence    = 1;
UserFaultInjection(idx).persistency = 0;
UserFaultInjection(idx).type        = 'STALEDATA';
```

The related fault STALEDATA is defined in defEquipmentFaults as:

```
% Stale data
Fault.Mag.STALEDATA.magFieldMeas_U    = [Id.FREEZE];
```

4.10 Documentation and Testing

The overall documentation is built by the following documentation sources

- DSC-Files for each single unit “<unitName>.dsc”
- Bus objectis description when using non-virtual buses
- DST-files for testing

5 Annex

5.1 Overview of Structural Models

The GAFE Structural Analysis contains the following structural models for AOCS sensors, actuators and analytical models (see also Figure 5-1):

AOCS Sensors:

- Accelerometer (ACC)
- Camera (CAM)
- Coarse Earth Sun Sensor (CESS
- Earth Sensor (ES)
- GNS Receiver (GNSR)
- LIDAR
- Magnetometer (MAG, 3-axes)
- Clock (OBT, absolute and relative)
- Rate Measurement Units (RMU, 3-axes)
- Sun Sensor (SS)
- Star Tracker (STR)

AOCS Actuators:

- Antenna Pointing Mechanism (APM)
- Magnetorquer (MTQ, 3-axes)
- Reaction Control System (RCS)
- Reaction Wheel (RW)
- Solar Array Drive (SADM)

Analytical Redundancy Models:

- Equations of Motion for Translation
- Equations of Motion for Rotation
- Earth Kinematics
- Earth Magnetic Field
- Earth Atmospheric Density
- Earth Direction
- Sun Direction
- Attitude Transformations
- Position Transformation

- Velocity Transformation
- Time Absolute/Relative
- Earth Gravity
- Non-Gravitational Acceleration
- Relative Position Direction Transformation

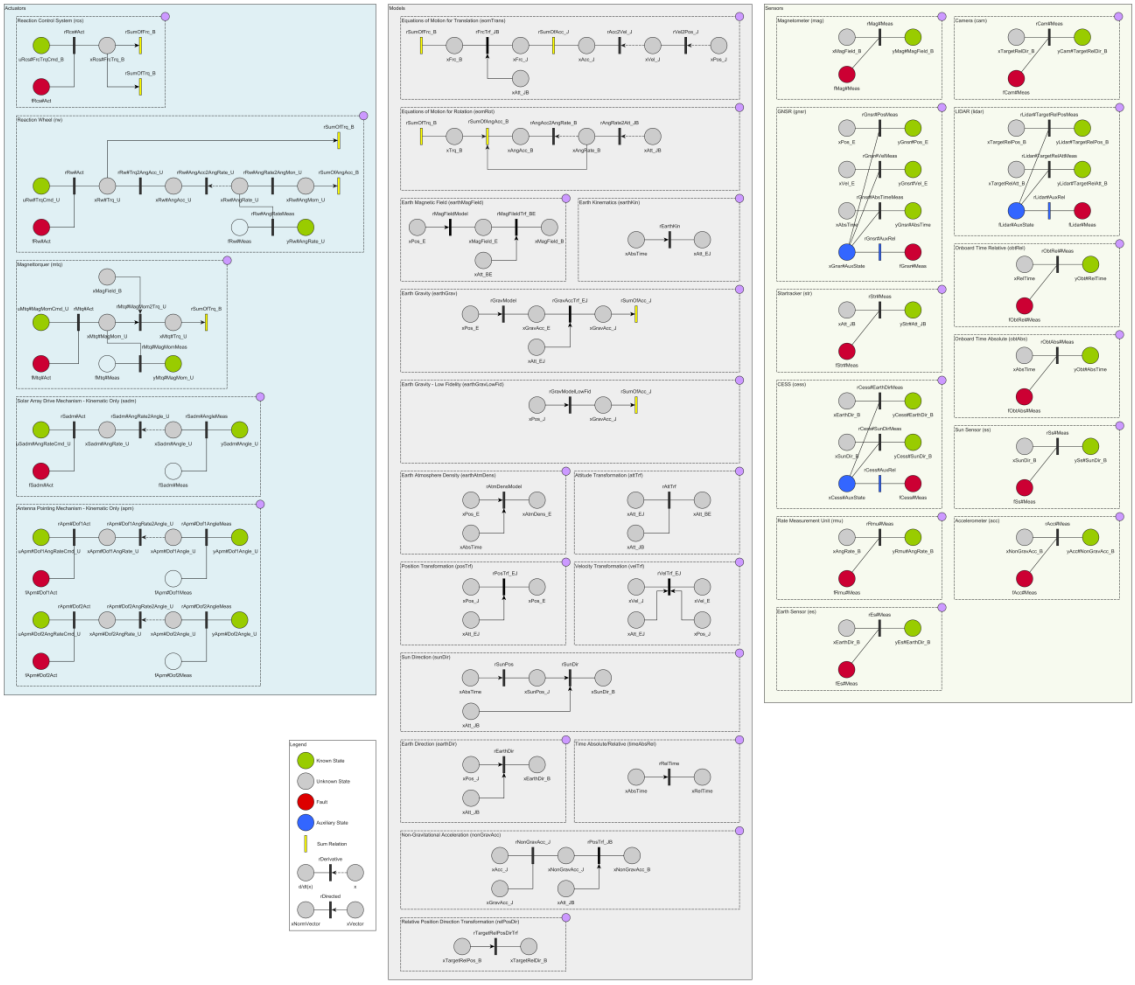


Figure 5-1 Overview of structural models contained in GAFE Structural Analysis (left: actuators, mid: analytical models, right: sensors).

The different types of states and non-standard relations follow the graphical conventions illustrated in Figure 3-3.

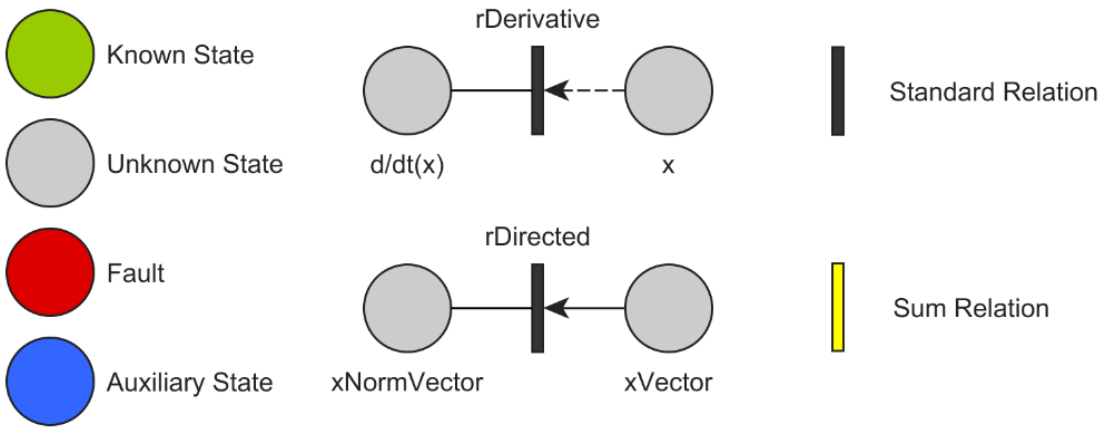


Figure 5-2: Overview of graphical conventions for states and relations.

5.1.1 Actuator Models

► Antenna Pointing Mechanism (apm)

A two degree of freedom antenna pointing mechanism (kinematic model).

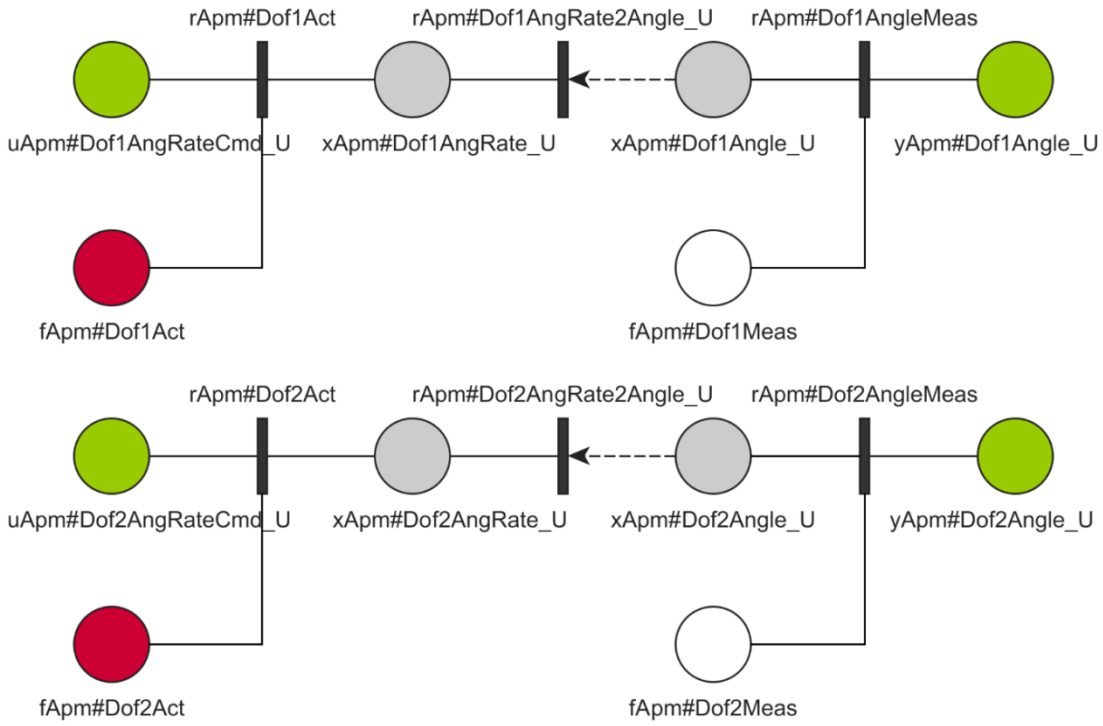


Figure 5-3: Structural Model of Antenna Pointing Mechanism.

► Magnetorquer (mtq)

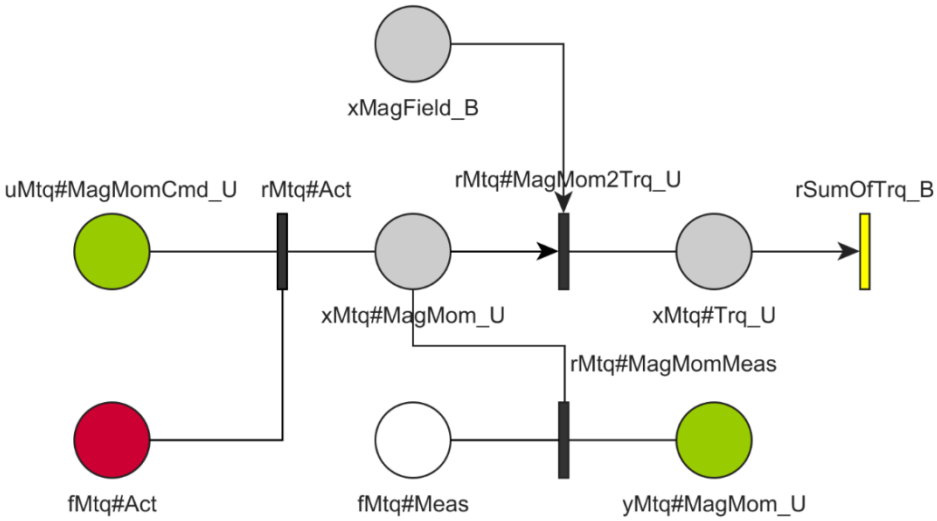


Figure 5-4: Structural Model of Magnetorquer.

► Reaction Control System (rcs)

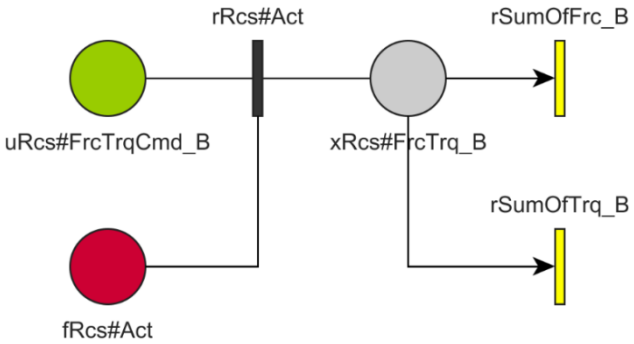


Figure 5-5: Structural Model of Reaction Control System.

► Reaction Wheel (rw)

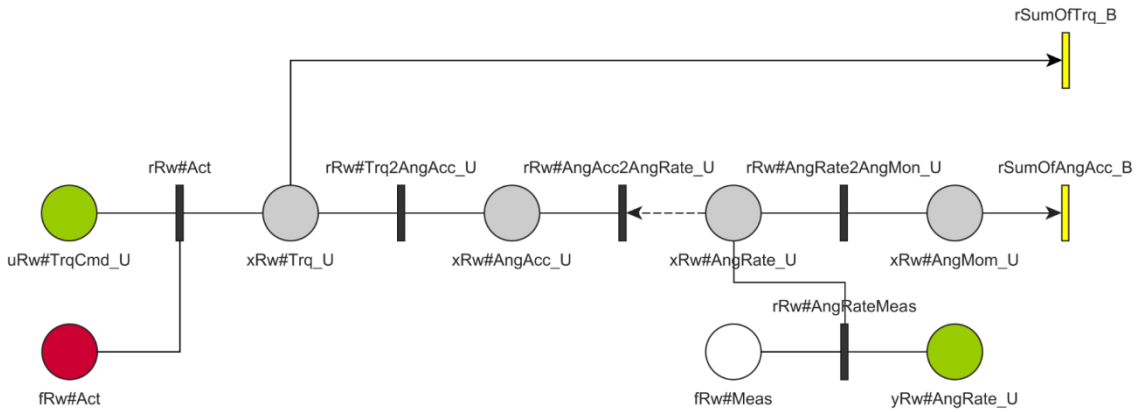


Figure 5-6: Structural Model of Reaction Wheel.

► Solar Array Drive Mechanism (sadm)

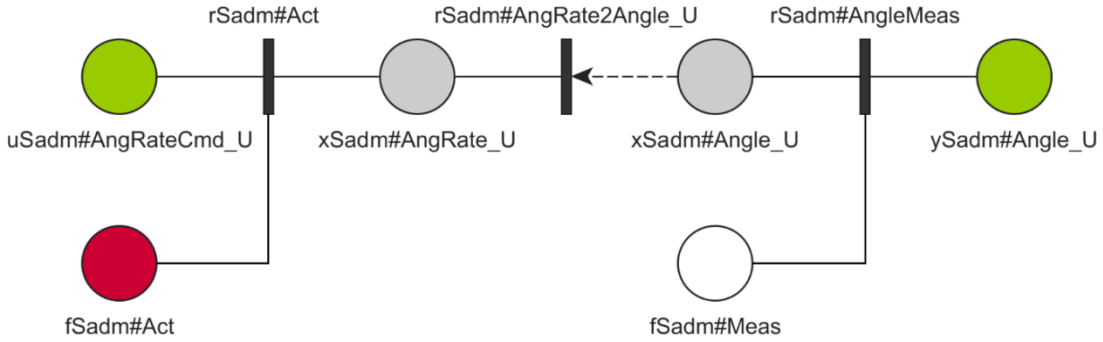


Figure 5-7: Structural Model of Solar Array Drive Mechanism.

5.1.2 Sensors Models

► Accelerometer (acc)

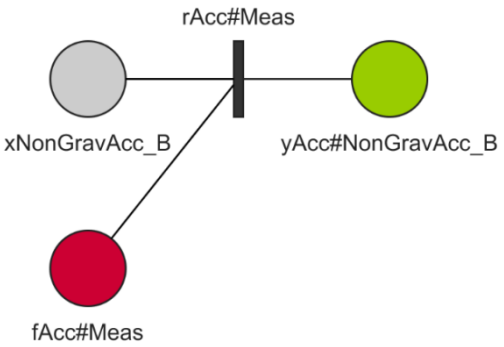


Figure 5-8: Structural Model of Accelerometer.

► Camera (cam

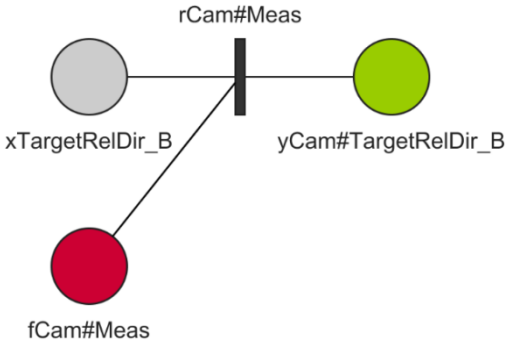


Figure 5-9: Structural Model of Camera.

► Es

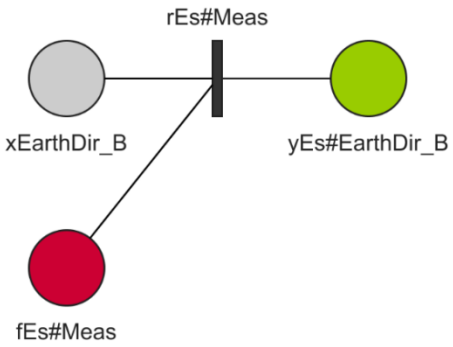


Figure 5-10: Structural Model of Earth Sensor.

► Coarse Earth Sun Sensor (cess)

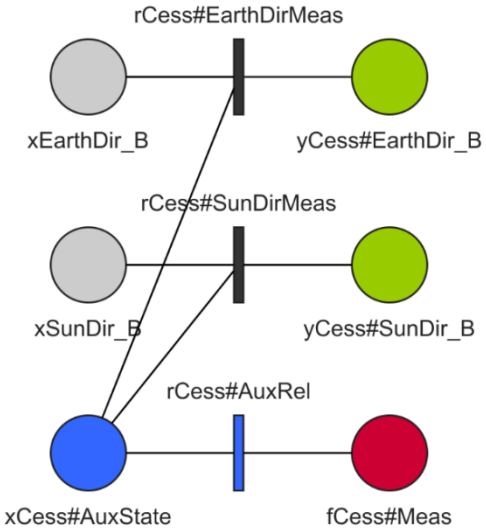


Figure 5-11: Structural Model of Coarse Earth Sun Sensor.

► Gnsr

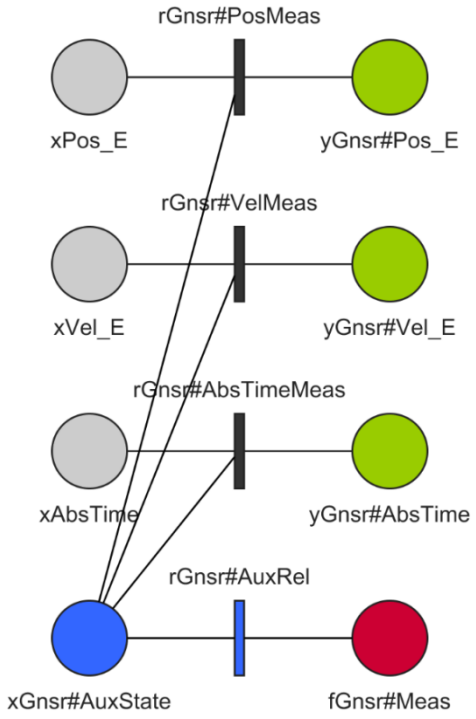


Figure 5-12: Structural Model of GNSS-Receiver.

► LIDAR (lidar)

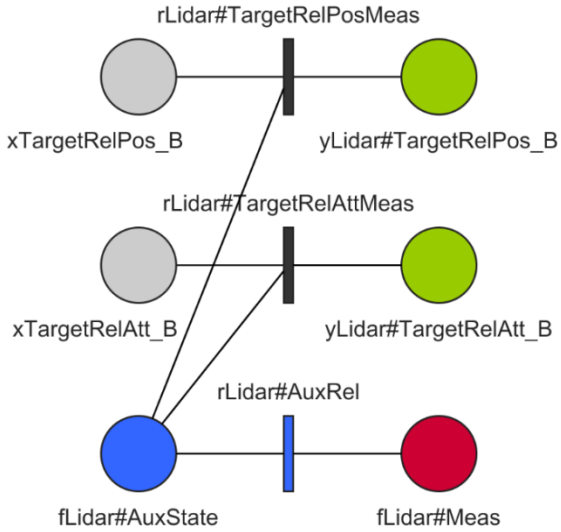


Figure 5-13: Structural Model of LIDAR.

► Magnetometer (mag)

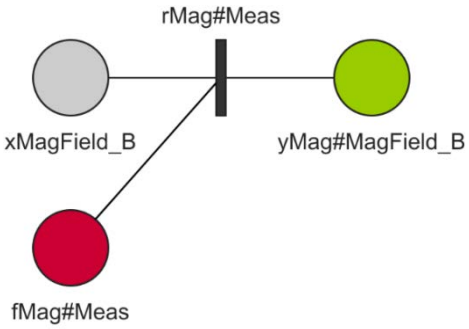


Figure 5-14: Structural Model of Magnetometer.

► *Onboard Clock Absolute Time (obtAbs)*

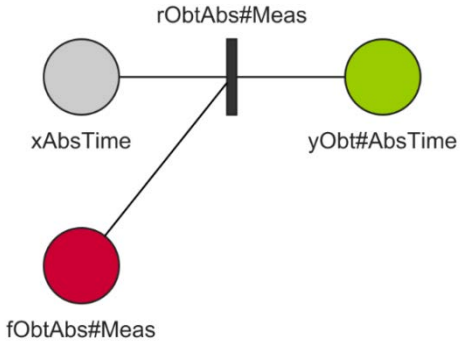


Figure 5-15: Structural Model of Onboard Clock Absolute Time.

► Onboard Clock Relative Time (obtRel)

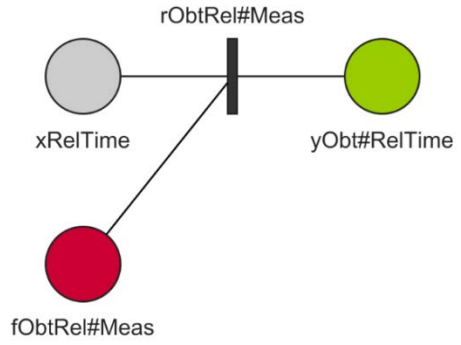


Figure 5-16: Structural Model of Onboard Clock Relative Time.

► Rate Measurement Unit (rmu)

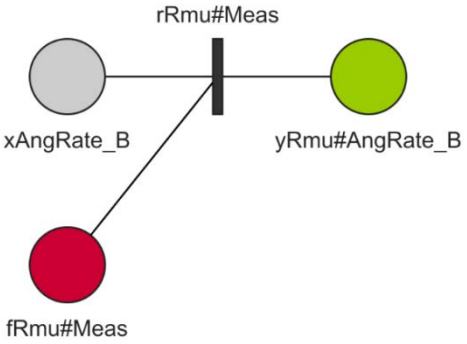


Figure 5-17: Structural Model of Rate Measurement Unit.

► Startracker (str)

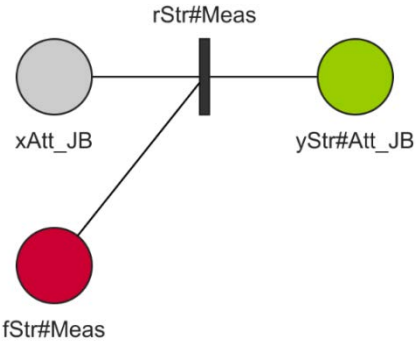


Figure 5-18: Structural Model of Startracker.

► Sun Sensor (ss)

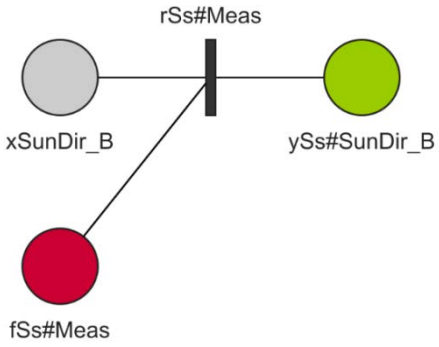


Figure 5-19: Structural Model of Sun Sensor.

5.1.3 Analytical Models

► Attitude Transformation (attTrf)

Attitude transformation between E, J, and B frame.

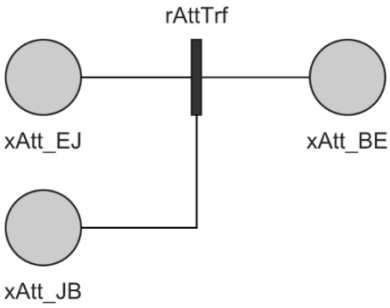


Figure 5-20: Structural Model of Attitude Transformation.

► Earth Atmosphere Density (earthAtmDens)

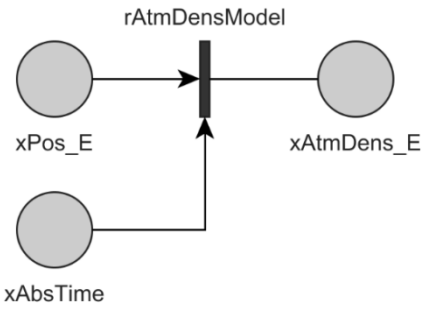


Figure 5-21: Structural Model of Earth Atmosphere Density.

► Earth Direction (earthDir)

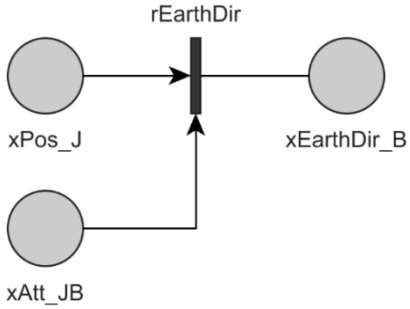


Figure 5-22: Structural Model of Earth Direction.

► Earth Gravity (earthGrav)

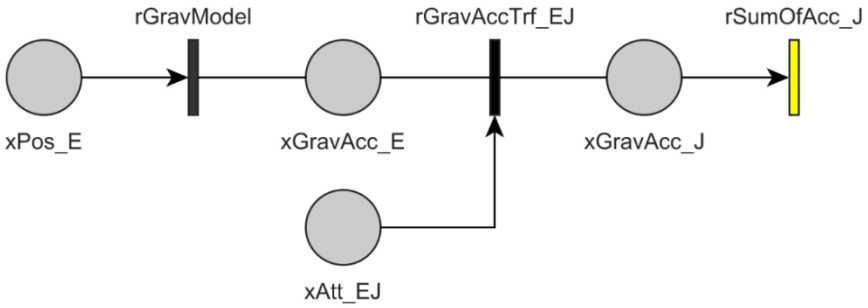


Figure 5-23: Structural Model of Earth Gravity.

► Earth Kinematics (earthKin)

Figure 5-24: Structural Model of Earth Kinematics.

► Earth Magnetic Field (earthMagField)

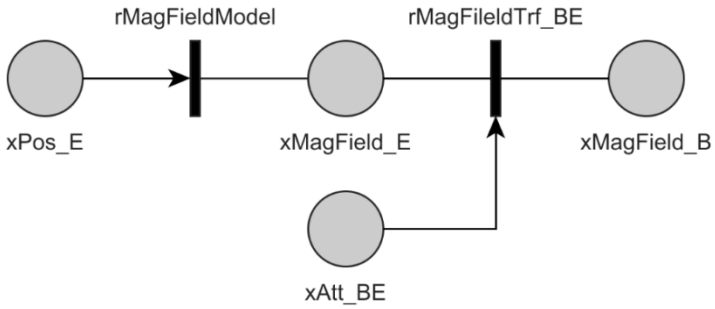


Figure 5-25: Structural Model of Earth Magnetic Field

► Equations of Motion for Rotation (eomRot)

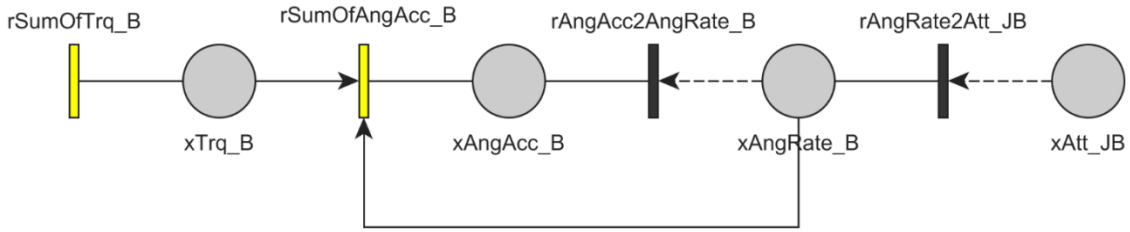


Figure 5-26: Structural Model of Equations of Motion for Rotation

Doc.No.: GAFE-UM-D7.5b
Issue: 2.1
Date: 17.08.2018

► Equations of Motion for Translation (eomTrans)

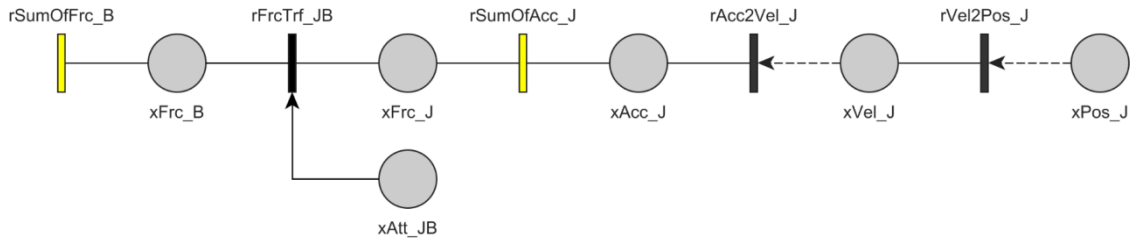


Figure 5-27: Structural Model of Equations of Motion for Translation.

► Non-Gravitational Acceleration (nonGravAcc)

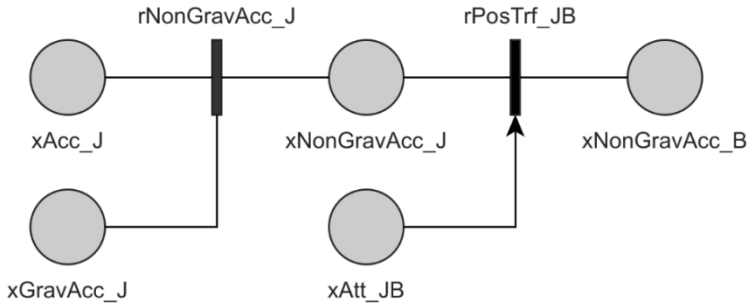


Figure 5-28: Structural Model of Non-Gravitational Acceleration.

► Position Transformation (posTrf)

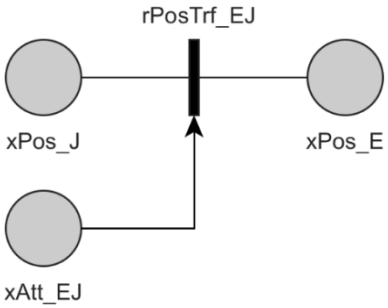


Figure 5-29: Structural Model of Position Transformation.

► Relative Position Direction Transformation (relPosDir)

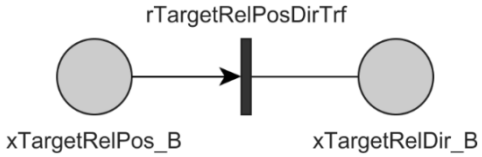


Figure 5-30: Structural Model of Relative Position Direction Transformation.

► Sun Direction (sunDir)

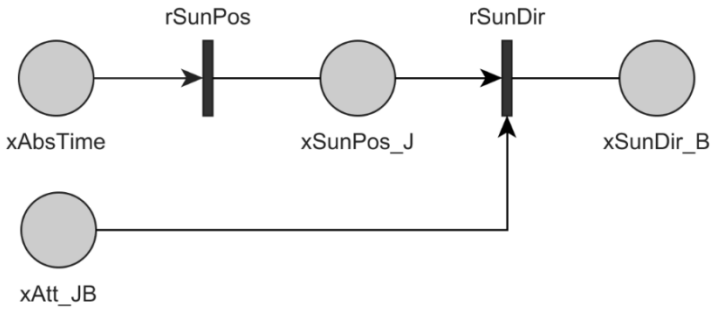


Figure 5-31: Structural Model of Sun Direction.

► Time Absolute/Relative (timeAbsRel)

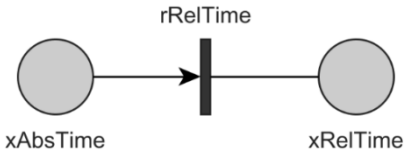


Figure 5-32: Structural Model of Time Absolute/Relative.

► Velocity Transformation (velTrf)

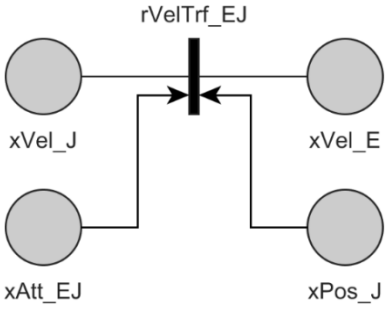


Figure 5-33: Structural Model of Velocity Transformation.

5.2 AOCS Algorithmic Components References

Find hereafter the AOCS Algorithmic Components, which the user may directly use or may take as template.

These are categorized into AOCS classes and listed within in alphabetical order.

5.2.1 Sensor Processing

5.2.1.1 camMeasProc

Camera measurement processing

The camera measurement is transformed to body frame.
Several checks are performed before processing: that Camera is communicating, has delivered new data to be processed, has declared its measurement as valid, does not deliver trash, is alive which means that it is changing it's value from one measurement to the other and that the norm of the measurement direction vector is "close" (specified limits) to 1.

Syntax:

```
[Data, Status, States, Params]= camMeasProc(1)
[Data, Status, States, Params]= camMeasProc(1, Params)
[Data, Status, States, Params]= camMeasProc(0, Params, CamHk, UnitsStatusExpected,
States)
```

High level inputs:

Component Name	Oneline Description
CamHk	Camera measurement (aka HK)
UnitsStatusExpected	System expected status of Unit

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
CamHk.relDirMeas_U	Relative direction measurement in unit frame	-	3	
CamHk.isMeasValid	Declared validity of measurement	-		
CamHk.isValidCnt	Validity counter	-		
UnitsStatusExpected.isCom	Unit communication status	-		defTrueFalseIds

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo	-		defAocsAlgoIsValidIds

Name	Oneline Description	Units	Size	Encoding
	component output			
Data.relDir_B	Processed relative direction measurement in body frame	-	3	
Data.relDist	Relative distance measurement (additional)	m	1	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isProcNewData	Flags new data is currently processed	-		defAocsAlgoIsValidIds
Status.isMeasDeclaredValid	Flags unit declared valid measurement	-		defAocsAlgoIsValidIds
Status.isFiniteData	Flags input data is not NaN an not INF	-		defAocsAlgoIsValidIds
Status.isLivingData	Flags data has changing values	-		defAocsAlgoIsValidIds
Status.isInLimRelDirNorm	Flags data being in expected range (RelDirNorm)	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.att_dcm_BUn	Unit alignment matrix	-	3,3	
Params.relDirNormEps	Epsilon for norm check of direction vector	-	1	
Params.filter.A	Filter discrete-time dynamic matrix		1	
Params.filter.B	Filter discrete-time input matrix		3	
Params.filter.C	Filter discrete-time output matrix		3	
Params.filter.D	Filter discrete-time feedthrough matrix		3,3	

Parameters Default Value:

Parameter Name	Parameter Default Value
att_dcm_BUn	[1, 0, 0; 0, 1, 0; 0, 0, 1]
relDirNormEps	1e-14
filter.signalDim	3
filter.A	0
filter.B	[0, 0, 0]

Parameter Name	Parameter Default Value
filter.C	[0; 0; 0]
filter.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]

States:

Name	Oneline Description	Units	Size	Encoding
States.isValidCnt	Validity counter	-		
States.relDirMeas_U	Relative direction measurement in unit frame		3	
States.filter.x	Filter state vector		1	
States.filter.u	Filter input vector		3	

5.2.1.2 clkMeasProc

Clock measurement processing

The onboard time OBT, typically distributed by the System, is treated as a sensor measurement thus underlying same mechanism and checks as any other sensor.
A user specifiable offset allows to adjust the time.
Several checks are performed before processing: that Clock is communicating, has delivered new data to be processed, does not deliver trash, is alive which means that it is changing it's value from one measurement to the other and within a specified range.

Syntax:

```
[Data, Status, States, Params]= clkMeasProc(1)
[Data, Status, States, Params]= clkMeasProc(1, Params)
[Data, Status, States, Params]= clkMeasProc(0, Params, ClkHk, UnitsStatusExpected,
States)
```

High level inputs:

Component Name	Oneline Description
ClkHk	OBC Clock measurement (aka HK)
UnitsStatusExpected	System expected status of Unit

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
ClkHk.isValidCnt	Validity counter of measurement (raw)	-		
ClkHk.obtTimeMeas	Measured onboard time OBT (raw)	s	1	
UnitsStatusExpected.isCom	Unit communication status	-	1	defTrueFalseIds

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.obtTime	Measured onboard time OBT	s	1	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isProcNewData	Flags new data is currently processed	-		defAocsAlgoIsValidIds
Status.isFiniteData	Flags input data is not NaN an not INF	-		defAocsAlgoIsValidIds

Name	Oneline Description	Units	Size	Encoding
Status.isLivingData	Flags data has changing values	-		defAocsAlgoIsValidIds
Status.isInLimObtTime	Flags data being in expected range (ObtTime)	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.obtTimeLim	Lower and upper limits for OBT range check	s	2	
Params.offset	Offset adjust vakue	s	1	

Parameters Default Value:

Parameter Name	Parameter Default Value
obtTimeLim	[1000000000, 2000000000]
offset	0

States:

Name	Oneline Description	Units	Size	Encoding
States.isValidCnt	Validity counter of measurement (raw)	-		
States.obtTimeMeas	Measured onboard time OBT (raw)	s	1	

5.2.1.3 esMeasProc

Earth Sensor measurement processing

Processes the measurement of one Earth Sensor by transforming it to body frame (B).
The measurement direction is checked that it is close to 1 within given limits.

Syntax:

```
[Data, Status, States, Params]= esMeasProc(1)
[Data, Status, States, Params]= esMeasProc(1, Params)
[Data, Status, States, Params]= esMeasProc(0, Params, EsHk, UnitsStatusExpected, States)
```

High level inputs:

Component Name	Oneline Description
EsHk	Earth Sensor measurement (aka HK)
UnitsStatusExpected	System expected status of Unit

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
EsHk.isValidCnt	Validity counter	-		
EsHk.satEarthDirMeas_U	Measured Earth direction	-	3	
UnitsStatusExpected.isCom	Unit communication status	-		defTrueFalseIds

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.earthDir_B	Processed measured Earth direction in body frame	-	3	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isProcNewData	Flags new data is currently processed	-		defAocsAlgoIsValidIds
Status.isFiniteData	Flags input data is not NaN an not INF	-		defAocsAlgoIsValidIds
Status.isLivingData	Flags data has changing values	-		defAocsAlgoIsValidIds
Status.isInLimEarthDirNorm	Flags data being in expected range (EarthDirNorm)	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.att_dcm_BUn	Unit alignment matrix	-	3,3	
Params.earthDirNormEps	Epsilon for norm check of direction vector	-	1	
Params.filter.A	Filter discrete-time dynamic matrix		1	
Params.filter.B	Filter discrete-time input matrix		3	
Params.filter.C	Filter discrete-time output matrix		3	
Params.filter.D	Filter discrete-time feedthrough matrix		3,3	

Parameters Default Value:

Parameter Name	Parameter Default Value
att_dcm_BUn	[1, 0, 0; 0, 1, 0; 0, 0, 1]
earthDirNormEps	1e-14
filter.signalDim	3
filter.A	0
filter.B	[0, 0, 0]
filter.C	[0; 0; 0]
filter.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]

States:

Name	Oneline Description	Units	Size	Encoding
States.isValidCnt	Validity counter of last measurement		1	
States.satEarthDirMeas_U	Last measured Earth direction		3	

5.2.1.4 gnsrMeasProc

GNSR measurement processing

Processes the measurement of the GNSR.
Several checks are performed before processing: that GNSR is communicating, has delivered new data to be processed, has declared its measurements as valid*, does not deliver trash, is alive which means that it is changing it's values (position and velocity) from one measurement to the other and that the norm of the measurement position and velocity vectors are within specified limits.
*Valid declaration is deduced from a combination of the current navigation mode and from the position and velocity quality indices which are checked to be lower than specified limits.

Syntax:

```
[Data, Status, States, Params]= gnsrMeasProc(1)
[Data, Status, States, Params]= gnsrMeasProc(1, Params)
[Data, Status, States, Params]= gnsrMeasProc(0, Params, GnsrHk, UnitsStatusExpected,
States)
```

High level inputs:

Component Name	Oneline Description
GnsrHk	GNSS Receiver measurement (aka HK)
UnitsStatusExpected	System expected status of Unit

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
GnsrHk.isValidCnt	Validity counter	-		
GnsrHk.satPosMeas_E	Measured satellite position (E)	m	3	
GnsrHk.satVelMeas_E	Measured satellite velocity (E)	m/s	3	
GnsrHk.gpsTimeMeas	GPS-time of measurement	days	1	
GnsrHk.posQualIndex	Position quality index	-	1	
GnsrHk.timeQualIndex	Time quality index	-	1	
GnsrHk.navigationMode	Navigation Mode	-	1	defGnsrNavModes
UnitsStatusExpected.isCom	Unit communication status	-		defTrueFalseIds

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo	-		defAocsAlgoIsValidIds

Name	Oneline Description	Units	Size	Encoding
	component output			
Data.satPos_E	Processed measured satellite position (E)	m	3	
Data.satVel_E	Processed measured satellite velocity (E)	m/s	3	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isProcNewData	Flags new data is currently processed	-		defAocsAlgoIsValidIds
Status.isMeasDeclaredValid	Flags unit declared valid measurement	-		defAocsAlgoIsValidIds
Status.isFiniteData	Flags input data is not NaN an not INF	-		defAocsAlgoIsValidIds
Status.isLivingData	Flags data has changing values	-		defAocsAlgoIsValidIds
Status.isInLimSatPosNorm	Flags data being in expected range (SatPosNorm)	-		defAocsAlgoIsValidIds
Status.isInLimSatVelNorm	Flags data being in expected range (SatVelNorm)	-		defAocsAlgoIsValidIds
Status.isInLimSatPosVelAngle	Flags data being in expected range (SatPosVelAngle)	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.satPosNormLim	Lower and upper limits of norm of position vector	m	2	
Params.satVelNormLim	Lower and upper limits of norm of velocity vector	m/s	2	
Params.posQualIndexLim	Upper limit of position quality index	-	1	
Params.timeQualIndexLim	Upper limit of time quality index	-	1	
Params.filterPos.A	Position-filter discrete-time dynamic matrix		1	
Params.filterPos.B	Position-filter discrete-time input matrix		3	
Params.filterPos.C	Position-filter discrete-time output matrix		3	
Params.filterPos.D	Position-filter discrete-time feedthrough matrix		3,3	

Name	Oneline Description	Units	Size	Encoding
Params.filterVel.A	Velocity-filter discrete-time dynamic matrix		1	
Params.filterVel.B	Velocity-filter discrete-time input matrix		3	
Params.filterVel.C	Velocity-filter discrete-time output matrix		3	
Params.filterVel.D	Velocity-filter discrete-time feedthrough matrix		3,3	

Parameters Default Value:

Parameter Name	Parameter Default Value
satPosNormLim	[6500000, 7500000]
satVelNormLim	[7300, 8000]
satPosVelAngleLim	0.08726646259971647
posQualIndexLim	10
timeQualIndexLim	10
filterPos.signalDim	3
filterPos.A	0
filterPos.B	[0, 0, 0]
filterPos.C	[0; 0; 0]
filterPos.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]
filterVel.signalDim	3
filterVel.A	0
filterVel.B	[0, 0, 0]
filterVel.C	[0; 0; 0]
filterVel.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]

States:

Name	Oneline Description	Units	Size	Encoding
States.isValidCnt	Validity counter	-		
States.satPosMeas_E	Measured satellite position (E)	m	3	
States.satVelMeas_E	Measured satellite velocity (E)	m/s	3	
States.filterPos.x	Filter state vector		1	
States.filterPos.u	Filter input vector		3	

Name	Oneline Description	Units	Size	Encoding
States.filterVel.x	Filter state vector		1	
States.filterVel.u	Filter input vector		3	

5.2.1.5 lidarMeasProc

Lidar measurement processing

The Lidar measurement is transformed to body frame.
Several checks are performed before processing: that Lidar is communicating, has delivered new data to be processed, has declared its measurement as valid, does not deliver trash, is alive which means that it is changing it's value from one measurement to the other and that the norm of the measurement position vector is within specified limits.

Syntax:

```
[Data, Status, States, Params]= lidarMeasProc(1)
[Data, Status, States, Params]= lidarMeasProc(1, Params)
[Data, Status, States, Params]= lidarMeasProc(0, Params, LidarHk, UnitsStatusExpected,
States)
```

High level inputs:

Component Name	Oneline Description
LidarHk	Lidar measurement (aka HK)
UnitsStatusExpected	System expected status of Unit

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
LidarHk.relPosMeas_U	Relative position measurement in unit frame	-	3	
LidarHk.isMeasValid	Declared validity of measurement	-		
LidarHk.isValidCnt	Validity counter	-		
UnitsStatusExpected.isCom	Unit communication status	-		defTrueFalseIds

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.relPos_B	Processed relative position measurement in body frame	m	3	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isProcNewData	Flags new data is currently processed	-		defAocsAlgoIsValidIds

Name	Oneline Description	Units	Size	Encoding
Status.isMeasDeclaredValid	Flags unit declared valid measurement	-		defAocsAlgoIsValidIds
Status.isFiniteData	Flags input data is not NaN an not INF	-		defAocsAlgoIsValidIds
Status.isLivingData	Flags data has changing values	-		defAocsAlgoIsValidIds
Status.isInLimRelPosNorm	Flags data being in expected range (RelPosNorm)	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.att_dcm_BUn	Unit alignment matrix	-	3,3	
Params.relPosNormLim	Lower and upper limits for relative position norm check	-	2	
Params.filter.A	Filter discrete-time dynamic matrix		1	
Params.filter.B	Filter discrete-time input matrix		3	
Params.filter.C	Filter discrete-time output matrix		3	
Params.filter.D	Filter discrete-time feedthrough matrix		3,3	

Parameters Default Value:

Parameter Name	Parameter Default Value
att_dcm_BUn	[1, 0, 0; 0, 1, 0; 0, 0, 1]
relPosNormLim	[1, 6000000]
filter.signalDim	3
filter.A	0
filter.B	[0, 0, 0]
filter.C	[0; 0; 0]
filter.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]

States:

Name	Oneline Description	Units	Size	Encoding
States.isValidCnt	Validity counter	-		

Name	Oneline Description	Units	Size	Encoding
States.relPosMeas_U	Relative position measurement in unit frame		3	
States.filter.x	Filter state vector		1	
States.filter.u	Filter input vector		3	

5.2.1.6 magMeasProc

Processes the measurements for one Magnetometer

The measurements are filtered and transformed to body frame applying a scale and a bias. Several checks are performed before processing: that Magnetometer is communicating, has delivered new data to be processed, does not deliver trash, is alive which means that it is changing it's value from one measurement to the other and within a specified range.

Syntax:

```
[Data, Status, States, Params]= magMeasProc(1)
[Data, Status, States, Params]= magMeasProc(1, Params)
[Data, Status, States, Params]= magMeasProc(0, Params, MagHk, UnitsStatusExp, States)
```

High level inputs:

Component Name	Oneline Description
MagHk	Magnetometer measurement (aka HK)
UnitsStatusExpected	System expected status of Unit

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
MagHk.isValidCnt	Validity counter of measurement	-	1	
MagHk.magFieldMeas_U	Measured magnetic field in body frame	T	1	
UnitsStatusExpected.isCom	Unit communication status	-	1	defTrueFalseIds

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.magField_B	Filtered measured magnetic field in body frame	T	3	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isProcNewData	Flags new data is currently processed	-		defAocsAlgoIsValidIds
Status.isFiniteData	Flags input data is not NaN an not INF	-		defAocsAlgoIsValidIds
Status.isLivingData	Flags data has changing	-		defAocsAlgoIsValidIds

Name	Oneline Description	Units	Size	Encoding
	values			
Status.isInLimMagFieldNorm	Flags data being in expected range (magFieldNorm)	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.att_dcm_BUn	Unit alignment matrix	-	3,3	
Params.magFieldNormLim	Lower and upper limits for norm check of magnetic field	T	2	
Params.scale_B	Scale correction factor	-	3	
Params.bias_B	Bias correction	T	3	
Params.filter.A	Filter discrete-time dynamic matrix		1	
Params.filter.B	Filter discrete-time input matrix		3	
Params.filter.C	Filter discrete-time output matrix		3	
Params.filter.D	Filter discrete-time feedthrough matrix		3,3	

Parameters Default Value:

Parameter Name	Parameter Default Value
att_dcm_BUn	[1, 0, 0; 0, 1, 0; 0, 0, 1]
magFieldNormLim	[0, 0.0001]
scale_B	[1; 1; 1]
bias_B	[0; 0; 0]
filter.signalDim	3
filter.A	0
filter.B	[0, 0, 0]
filter.C	[0; 0; 0]
filter.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]

States:

Name	Oneline Description	Units	Size	Encoding
States.isValidCnt	Validity counter of last measurement		1	

Name	Oneline Description	Units	Size	Encoding
States.filter.x	Filter state vector		1	
States.filter.u	Filter input vector		3	

5.2.1.7 rmuMeasProc

Rate Measurement Unit measurement processing

Processes the measurement of one Rate Measurement Unit by transforming it to body frame (B) after applying a filter.
The norm of the measured rate is checked to be within specified limits.

Syntax:

```
[Data, Status, States, Params]= rmuMeasProc(1)
[Data, Status, States, Params]= rmuMeasProc(1, Params)
[Data, Status, States, Params]= rmuMeasProc(0, Params, RmuHk, UnitsStatusExpected,
States)
```

High level inputs:

Component Name	Oneline Description
RmuHk	Rate Measurement Unit measurement (aka HK)
UnitsStatusExpected	System expected status of Unit

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
RmuHk.isValidCnt	Validity counter	-		
RmuHk.satRateMeas_U	Measured satellite rate in unit frame	rad/s	3	
UnitsStatusExpected.isCom	Unit communication status	-		

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.satRate_B	Processed measured satellite rate in body frame	rad/s	3	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isProcNewData	Flags new data is currently processed	-		defAocsAlgoIsValidIds
Status.isFiniteData	Flags input data is not NaN an not INF	-		defAocsAlgoIsValidIds
Status.isLivingData	Flags data has changing values	-		defAocsAlgoIsValidIds

Name	Oneline Description	Units	Size	Encoding
Status.isInLimSatRateNorm	Flags data being in expected range (SatRateNorm)	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.att_dcm_BUn	Unit alignment matrix	-	3,3	
Params.satRateNormLim	Lower and upper limits of norm of satellite rate	rad/s	2	
Params.filter.A	Filter discrete-time dynamic matrix		1	
Params.filter.B	Filter discrete-time input matrix		3	
Params.filter.C	Filter discrete-time output matrix		3	
Params.filter.D	Filter discrete-time feedthrough matrix		3,3	

Parameters Default Value:

Parameter Name	Parameter Default Value
att_dcm_BUn	[1, 0, 0; 0, 1, 0; 0, 0, 1]
satRateNormLim	[0, 0.174532925199433]
filter.signalDim	3
filter.A	0
filter.B	[0, 0, 0]
filter.C	[0; 0; 0]
filter.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]

States:

Name	Oneline Description	Units	Size	Encoding
States.isValidCnt	Validity counter of last measurement	-	1	
States.satRateMeas_U	Last measured satellite rate	rad/s	3	
States.filter.x	Filter state vector		1	
States.filter.u	Filter input vector		3	

5.2.1.8 rwMeasProc

Reaction Wheel measurement processing

Checks for spikes and applies a filter on the measurement. In case no valid measurement could be acquired the last valid rate is used for a limited number of steps. The output rate is flagged valid after settable number of valid measurements acquired and is reset to invalid after a settable number of invalid measurements. Finally the wheel angular momentum is computed.

Syntax:

```
[Data, Status, States, Params]= rwMeasProc(1)
[Data, Status, States, Params]= rwMeasProc(1, Params)
[Data, Status, States, Params]= rwMeasProc(0, Params, RwHk, UnitsStatusExpected, States)
```

High level inputs:

Component Name	Oneline Description
RwHk	Reaction Wheel measurement (aka HK)
UnitsStatusExpected	System expected status of Unit

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
RwHk.isValidCnt	Validity counter	-		
RwHk.rwRateMeas	Measured RW rate	rad/s	1	
UnitsStatusExpected.isCom	Unit communication status	-		defTrueFalseIds

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.rwRate	Processed measured RW rate	rad/s	1	
Data.rwAngMom	RW measured angular momentum	Nms	1	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isProcNewData	Flags new data is currently processed	-		defAocsAlgoIsValidIds
Status.isFiniteData	Flags input data is not NaN an not INF	-		defAocsAlgoIsValidIds

Name	Oneline Description	Units	Size	Encoding
Status.isLivingData	Flags data has changing values	-		defAocsAlgoIsValidIds
Status.isInLimRwRate	Flags data being in expected range (RwRate)	-		defAocsAlgoIsValidIds
Status.isInLimRwRateChange	Flags data being in expected range (RwRateChange)	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.rwMoi	RW moment of inertia	kgm2	1	
Params.rwRateLim	Maximum rate limit	rad/s	1	
Params.rwRateChangeLim	Limit for change of rate (for spikes)	rad/s	1	
Params.isNotInLimRwRateChangeCntLim	Allowed number not to be in limit	-	1	
Params.filter.A	Filter discrete-time dynamic matrix			
Params.filter.B	Filter discrete-time input matrix			
Params.filter.C	Filter discrete-time output matrix			
Params.filter.D	Filter discrete-time feedthrough matrix			

Parameters Default Value:

Parameter Name	Parameter Default Value
rwMoi	1
rwRateLim	209.4395102393196
rwRateChangeLim	1
isNotInLimRwRateChangeCntLim	2
filter.signalDim	1
filter.A	0
filter.B	0
filter.C	0
filter.D	1

States:

Name	Oneline Description	Units	Size	Encoding
States.isValidCnt	Validity counter	-		
States.rwRateMeas	Measured RW rate	rad/s	1	
States.rwRate	Processed measured RW rate	rad/s	1	
States.isNotInLimRwRateChangeCnt	Counter for allowed number not to be in limit	-	1	
States.isLivingDataBuffer	Data buffer to detect not living data	-		
States.filter.x	Filter state vector			
States.filter.u	Filter input vector			

5.2.1.9 ssMeasProc

Sun Sensor measurement processing

Processes the measurement of one Sun Sensor by transforming it to body frame (B).
If two consecutive measurements are valid the Sun rate is computed.
Sun direction and Sun rate are filtered.
The measurement direction is checked that it is close to 1 within given limits.

Syntax:

```
[Data, Status, States, Params]= ssMeasProc(1)
[Data, Status, States, Params]= ssMeasProc(1, Params)
[Data, Status, States, Params]= ssMeasProc(0, Params, SsHk, UnitsStatusExpected, States)
```

High level inputs:

Component Name	Oneline Description
SsHk	Sun Sensor measurement (aka HK)
UnitsStatusExpected	System expected status of Unit

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
SsHk.isValidCnt	Validity counter	-		
SsHk.satSunDirMeas_U	Measured Sun direction	-	3	
UnitsStatusExpected.isCom	Unit communication status	-		

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.isValidSunDir	Validity flag for Sun direction component output	-		defAocsAlgoIsValidIds
Data.isValidSunRate	Validity flag for Sun rate component output	-		defAocsAlgoIsValidIds
Data.sunDir_B	Processed measured Sun direction in body frame	-	3	
Data.sunRate_B	Computed Sun rate in body frame	rad/s	3	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds

Name	Oneline Description	Units	Size	Encoding
Status.isValidSunDir	Validity flag for Sun direction component output	-		defAocsAlgoIsValidIds
Status.isValidSunRate	Validity flag for Sun rate component output	-		defAocsAlgoIsValidIds
Status.isProcNewData	Flags new data is currently processed	-		defAocsAlgoIsValidIds
Status.isFiniteData	Flags input data is not NaN an not INF	-		defAocsAlgoIsValidIds
Status.isLivingData	Flags data has changing values	-		defAocsAlgoIsValidIds
Status.isInLimSunDirNorm	Flags data being in expected range (SunDirNorm)	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.att_dcm_BUn	Unit alignment matrix	-	3,3	
Params.sunDirNormEps	Epsilon for norm check of direction vector	-	1	
Params.sampleTime	Sample time required for Sun rate computation	s	1	
Params.filterSunDir.A	Sun direction filter discrete-time dynamic matrix		1	
Params.filterSunDir.B	Sun direction filter discrete-time input matrix		3	
Params.filterSunDir.C	Sun direction filter discrete-time output matrix		3	
Params.filterSunDir.D	Sun direction filter discrete-time feedthrough matrix		3,3	
Params.filterSunRate.A	Sun rate filter discrete-time dynamic matrix		1	
Params.filterSunRate.B	Sun rate filter discrete-time input matrix		3	
Params.filterSunRate.C	Sun rate filter discrete-time output matrix		3	
Params.filterSunRate.D	Sun rate filter discrete-time feedthrough matrix		3,3	

Parameters Default Value:

Parameter Name	Parameter Default Value
att_dcm_BUn	[1, 0, 0; 0, 1, 0; 0, 0, 1]
sunDirNormEps	1e-14
sampleTime	0.5

Parameter Name	Parameter Default Value
filterSunDir.signalDim	3
filterSunDir.A	0
filterSunDir.B	[0, 0, 0]
filterSunDir.C	[0; 0; 0]
filterSunDir.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]
filterSunRate.signalDim	3
filterSunRate.A	0
filterSunRate.B	[0, 0, 0]
filterSunRate.C	[0; 0; 0]
filterSunRate.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]

States:

Name	Oneline Description	Units	Size	Encoding
States.isValidCnt	Validity counter of last measurement		1	
States.satSunDirMeas_U	Last measured Sun direction		3	
States.initSunRateComputation	Flags if Sun rate computation requires initialization	-		
States.filterSunDir.x	Sun direction filter state vector		1	
States.filterSunDir.u	Sun direction filter input vector		3	
States.filterSunRate.x	Sun rate filter state vector		1	
States.filterSunRate.u	Sun rate filter input vector		3	

5.2.1.10 strMeasProc

Startracker measurement processing

The attitude measurement is processed and converted such that it outputs the spacecraft inertial attitude and rate.
The inertial rate is computed from two consecutive attitude unprocessed measurements, computing the delta attitude and dividing by the time difference. This is only done if the time difference is smaller than a specified limit. The rate is filtered.
Several further checks are performed before processing: that Startracker is communicating, has delivered new data to be processed, has declared its measurements as valid (deduced from the quality index), does not deliver trash, is alive which means that it is changing it's values (position and velocity) from one measurement to the other and that the norm of the measurement position and velocity vectors are within specified limits.
Since this component delivers two outputs, two respective validity information are provided. The overall validity flag will either be NOK, PART, or OK.

Syntax:

```
[Data, Status, States, Params]= strMeasProc(1)
[Data, Status, States, Params]= strMeasProc(1, Params)
[Data, Status, States, Params]= strMeasProc(0, Params, StrHk, UnitsStatusExpected,
States)
```

High level inputs:

Component Name	Oneline Description
StrHk	Star Tracker measurement (aka HK)
UnitsStatusExpected	System expected status of Unit

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
StrHk.isValidCnt	Validity counter	-		
StrHk.satAttMeas_qUJ	Measurement quaternion	-	4	
StrHk.satAttMeasCoiTime	Measurement timestamp (center of integration)	days	1	
StrHk.satAttQualIndex	Measurement quality index	-		defEquipmentQualityIds
UnitsStatusExpected.isCom	Unit communication status	-		defTrueFalseIds

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds

Name	Oneline Description	Units	Size	Encoding
Data.isValidAtt	Validity flag for attitude output	-		defAocsAlgoIsValidIds
Data.isValidRate	Validity flag for rate output	-		defAocsAlgoIsValidIds
Data.satAtt_dcm_BJ	Processed measured attitude	-	3,3	
Data.satRate_B	Computed satellite rate (B)	rad/s	3	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isValidAtt	Validity flag for attitude output	-		defAocsAlgoIsValidIds
Status.isValidRate	Validity flag for rate output	-		defAocsAlgoIsValidIds
Status.isProcNewData	Flags new data is currently processed	-		defAocsAlgoIsValidIds
Status.isMeasDeclaredValid	Flags unit declared valid measurement	-		defAocsAlgoIsValidIds
Status.isFiniteData	Flags input data is not NaN an not INF	-		defAocsAlgoIsValidIds
Status.isLivingData	Flags data has changing values	-		defAocsAlgoIsValidIds
Status.isInLimSatAttQuatNorm	Flags data being in expected range (SatAttQuatNorm)	-		defAocsAlgoIsValidIds
Status.isInLimTimeDif	Flags data being in expected range (TimeDif)	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.att_dcm_BUn	Unit alignment matrix	-	3,3	
Params.satAttQuatNormEps	Epsilon for norm check of quaternion	-	1	
Params.filterAtt.A	Attitude filter discrete-time dynamic matrix		1	
Params.filterAtt.B	Attitude filter discrete-time input matrix		3	
Params.filterAtt.C	Attitude filter discrete-time output matrix		3	
Params.filterAtt.D	Attitude filter discrete-time feedthrough matrix		3,3	
Params.filterRate.A	Rate filter discrete-time dynamic matrix		1	
Params.filterRate.B	Rate filter discrete-time input matrix		3	

Name	Oneline Description	Units	Size	Encoding
Params.filterRate.C	Rate filter discrete-time output matrix		3	
Params.filterRate.D	Rate filter discrete-time feedthrough matrix		3,3	

Parameters Default Value:

Parameter Name	Parameter Default Value
att_dcm_BUn	[1, 0, 0; 0, 1, 0; 0, 0, 1]
satAttQuatNormEps	1e-09
timeDifLim	1
filterAtt.signalDim	3
filterAtt.A	0
filterAtt.B	[0, 0, 0]
filterAtt.C	[0; 0; 0]
filterAtt.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]
filterRate.signalDim	3
filterRate.A	0
filterRate.B	[0, 0, 0]
filterRate.C	[0; 0; 0]
filterRate.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]

States:

Name	Oneline Description	Units	Size	Encoding
States.isValidCnt	Validity counter	-		
States.satAttMeas_qUJ	Measurement quaternion		4	
States.satAttMeasCoiTime	Measurement timestamp (center of integration)	days	1	
States.filterAtt.x	Attitude filter state vector		1	
States.filterAtt.u	Attitude filter input vector		3	
States.filterRate.x	Rate filter state vector		1	
States.filterRate.u	Rate filter input vector		3	

5.2.2 Determination

5.2.2.1 ephemerisDet

Ephemeris Determination

Using the time as input, the rotation Matrix of the Earth and the Sun position are calculated using well known approximation formulas.
Nutation and precession are considered using IAU 1980 Nutation parameters. Nutation and precession can be parametrized on/off individually. Leap seconds and the delta UT1 to UTC can be specified as parameters.

Syntax:

```
[Data, Status, States, Params]= ephemerisDet(1)
[Data, Status, States, Params]= ephemerisDet(1, Params)
[Data, Status, States, Params]= ephemerisDet(0, Params, TimeEst, States)
```

High level inputs:

Component Name	Oneline Description
TimeEst	Estimated time

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
TimeEst.isValid	Validity flag for item "TimeEst"	-		defAocsAlgoIsValidIds
TimeEst.time	Estimated time	s	1	

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.dcm_JE	Earth rotation matrix	-	3,3	
Data.sunDir_J	Apparent Sun direction in inertial frame (J)	-	3	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.difTaiUtc	Difference between TAI and UTC	s	1	

Name	Oneline Description	Units	Size	Encoding
Params.difUt1Utc	Difference between UT1 and UTC	s	1	
Params.enaPrecession	Enable precession computation	-	1	
Params.enaNutation	Enable nutation computation	-	1	
Params.nutationParams	Nutation parameters IAU1980 (don't change)	-	10,9	

Parameters Default Value:

Parameter Name	Parameter Default Value
difTaiUtc	36
difUt1Utc	0
enaPrecession	0
enaNutation	0
nutationParams	[0, 0, 0, 0, 1, -171996, -174.2, 92025, 8.9; 0, 0, 2, -2, 2, -13187, -1.6, 5736, -3.1; 0, 0, 2, 0, 2, -2274, -0.2, 977, -0.5; 0, 0, 0, 0, 2, 2062, 0.2, -895, 0.5; 0, 1, 0, 0, 0, 1426, -3.4, 54, -0.1; 1, 0, 0, 0, 0, 712, 0.1, -7, 0; 0, 1, 2, -2, 2, -517, 1.2, 224, -0.6; 0, 0, 2, 0, 1, -386, -0.4, 200, 0; 1, 0, 2, 0, 2, -301, 0, 129, -0.1; 0, -1, 2, -2, 2, 217, -0.5, -95, 0.3]

States:

Name	Oneline Description	Units	Size	Encoding
States				

5.2.2.2 esEarthDirEst

Estimates the Earth direction based on Earth Sensor

Two methods are offered to estimate the Earth direction: Mean value computation and middle value computation of each component.
This is done based on the validity of each measurement and its usability.
The Earth rate is computed from two consecutive estimated Earth directions.

Syntax:

```
[Data, Status, States, Params]= esEarthDirEst(1)
[Data, Status, States, Params]= esEarthDirEst(1, Params)
[Data, Status, States, Params]= esEarthDirEst(0, Params, EsMeasProc,
UnitsStatusExpected, States)
```

High level inputs:

Component Name	Oneline Description
EsMeasProc	Processed Earth Sensor measurement
UnitsStatusExpected	System expected status of Unit

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
EsMeasProc.isValid	Validity flag for item "EsMeasProc"	-		defAocsAlgoIsValidIds
EsMeasProc.earthDir_B	Measured Earth direction in body frame	-	3	
UnitsStatusExpected.isUsbl	Unit usability status	-		defTrueFalseIds

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.isValidEarthDir	Validity flag for Earth direction component output	-		defAocsAlgoIsValidIds
Data.isValidEarthRate	Validity flag for Earth rate component output	-		defAocsAlgoIsValidIds
Data.earthDir_B	Estimated Earth direction in body frame	-	3	
Data.earthRate_B	Estimated Earth rate in body frame	rad/s	3	

Name	Oneline Description	Units	Size	Encoding
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isCrossCheckOk	Flags cross check results	-	numUnits	defAocsAlgoIsValidIds
Status.isUsblAny	Flags usability of least one unit	-		defAocsAlgoIsValidIds
Status.isEarthDetected	Flags if Earth is detected (i.e. any valid measurement)	-	1	
Status.isValidEarthDir	Validity flag for Earth direction component output	-		defAocsAlgoIsValidIds
Status.isValidEarthRate	Validity flag for Earth rate component output	-		defAocsAlgoIsValidIds
Status.isUsed	Flags if unit is used	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.numUnits	Number of units	-		
Params.method	Estimation method (2:mean,3:middle)	-	1	
Params.sampleTime	Sample time required for Earth rate computation	s	1	
Params.crossCheckAngLim	Earth direction cross check angle limit	rad	1	
Params.filterEarthDir.A	Earth direction filter discrete-time dynamic matrix		1	
Params.filterEarthDir.B	Earth direction filter discrete-time input matrix		3	
Params.filterEarthDir.C	Earth direction filter discrete-time output matrix		3	
Params.filterEarthDir.D	Earth direction filter discrete-time feedthrough matrix		3,3	
Params.filterEarthRate.A	Earth rate filter discrete-time dynamic matrix		1	
Params.filterEarthRate.B	Earth rate filter discrete-time input matrix		3	
Params.filterEarthRate.C	Earth rate filter discrete-time output matrix		3	
Params.filterEarthRate.D	Earth rate filter discrete-time feedthrough matrix		3,3	

Parameters Default Value:

Parameter Name	Parameter Default Value
numUnits	1
method	2
sampleTime	0.5
crossCheckAngLim	0.5235987755982988
filterEarthDir.signalDim	3
filterEarthDir.A	0
filterEarthDir.B	[0, 0, 0]
filterEarthDir.C	[0; 0; 0]
filterEarthDir.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]
filterEarthRate.signalDim	3
filterEarthRate.A	0
filterEarthRate.B	[0, 0, 0]
filterEarthRate.C	[0; 0; 0]
filterEarthRate.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]

States:

Name	Oneline Description	Units	Size	Encoding
States.earthDir_B	Last estimated Earth direction (B)	-	3	
States.initEarthRateComputation	Flags if Earth rate computation requires initialization	-		
States.filterEarthDir.x	Earth direction filter state vector		1	
States.filterEarthDir.u	Earth direction filter input vector		3	
States.filterEarthRate.x	Earth rate filter state vector		1	
States.filterEarthRate.u	Earth rate filter input vector		3	

5.2.2.3 magFieldEst

Estimates magnetic field and magnetic rate

The estimated magnetic field is chosen from the incoming Magnetometer measurements. Two estimation methods are offered.
Selection according to a user specified selection sequence, or middle value computation of each component. Each respecting the validity of the measurements and their usability status.
A cross check of all Magnetometer measurements is made and the ones which fail all checks against others discarded from further use.
In addition the magnetic rate is computed by pseudo differentiation of the estimated magnetic field.
The input expected to match the output of "magMeasProc" given as structure array with the number of processed Magnetometer units ("numUnits").
Since this component delivers two outputs, two respective validity information are provided. The overall validity flag will either be NOK, PART, or OK.

Syntax:

```
[Data, Status, States, Params]= magFieldEst(1)
[Data, Status, States, Params]= magFieldEst(1, Params)
[Data, Status, States, Params]= magFieldEst(0, Params, MagMeasProc, UnitsStatusExpected,
States)
```

High level inputs:

Component Name	Oneline Description
MagMeasProc	Processed Magnetometer measurements
UnitsStatusExpected	System expected status of Unit

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
MagMeasProc.isValid	Validity flag for item "MagMeasProc"	-		defAocsAlgoIsValidIds
MagMeasProc.magField_B	Processed measured magnetic field (body frame)	T	3	
UnitsStatusExpected.isUsbl	Unit usability status	-	numUnits	defTrueFalseIds

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.isValidMagField	Validity flag for magnetic field	-		defAocsAlgoIsValidIds

Name	Oneline Description	Units	Size	Encoding
	output			
Data.isValidMagRate	Validity flag for magnetic rate output	-		defAocsAlgoIsValidIds
Data.magField_B	Estimated magnetic field (body frame)	T	3	
Data.magRate_B	Estimated magnetic rate	rad/s	3	
Data.selId	Selected magnetic field measurement	-	1	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isValidMagField	Validity flag for magnetic field output	-		defAocsAlgoIsValidIds
Status.isValidMagRate	Validity flag for magnetic rate output	-		defAocsAlgoIsValidIds
Status.isCrossCheckOk	Flags cross check results	-	numUnits	defAocsAlgoIsValidIds
Status.isUsblAny	Flags usability of least one unit	-		defAocsAlgoIsValidIds
Status.isMagRateLow	Flags magnetic rate low condition	-		defAocsAlgoIsValidIds
Status.isUsed	Flags if unit is used	-	numUnits	defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.numUnits	Number of units	-		
Params.method	Estimation method (1:selection,2:middle)	-		
Params.selSeq	Selection sequence	-		
Params.rateLowLim	Upper limit for rate low condition check	rad/s	1	
Params.crossCheckNormLim	Magnetic field norm cross check limit	T	1	
Params.pdiffParams	Pseudo differentiation parameter		5	
Params.filterMagField.A	Magnetic field filter discrete-time dynamic matrix		1	
Params.filterMagField.B	Magnetic field filter discrete-time input matrix		3	
Params.filterMagField.C	Magnetic field filter discrete-time output matrix		3	
Params.filterMagField.D	Magnetic field filter discrete-time feedthrough matrix		3,3	

Name	Oneline Description	Units	Size	Encoding
Params.filterMagRate.A	Magnetic rate filter discrete-time dynamic matrix		1	
Params.filterMagRate.B	Magnetic rate filter discrete-time input matrix		3	
Params.filterMagRate.C	Magnetic rate filter discrete-time output matrix		3	
Params.filterMagRate.D	Magnetic rate filter discrete-time feedthrough matrix		3,3	

Parameters Default Value:

Parameter Name	Parameter Default Value
numUnits	1
method	1
selSeq	1
pdiffParams	[0.3333333333333333, 0.5, -0.8888888888888888, 0.6666666666666666, 0.7499999999999999]
rateLowLim	0.0174532925199433
crossCheckNormLim	1e-06
filterMagField.signalDim	3
filterMagField.A	0
filterMagField.B	[0, 0, 0]
filterMagField.C	[0; 0; 0]
filterMagField.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]
filterMagRate.signalDim	3
filterMagRate.A	0
filterMagRate.B	[0, 0, 0]
filterMagRate.C	[0; 0; 0]
filterMagRate.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]

States:

Name	Oneline Description	Units	Size	Encoding
States.pdiff	Pseudo differentiation state		3	
States.initPdiff	Pseudo differentiation initialisation flag		1	
States.filterMagField.x	Magnetic field filter state vector		1	

Name	Oneline Description	Units	Size	Encoding
States.filterMagField.u	Magnetic field filter input vector		3	
States.filterMagRate.x	Magnetic rate filter state vector		1	
States.filterMagRate.u	Magnetic rate filter input vector		3	

5.2.2.4 oop

Onboard Orbit Propagator

Determines the position and velocity of the satellite in inertial frame (J).
This is done by transforming the GNSR measurements from Earth (E) frame to (J). Therefore input "dcm_JE" from the Ephemeris determination (EphemerisDet) is used
Or if GNSR measurement is not valid or not usable by propagating autonomously to the current time. The propagation is performed using the RK4 algorithm taking the additional gravitational terms J2,J3,J4 into account.
Propagation will only be performed if at least one valid measurement of GNSR was acquired so far.
In both cases it is required that the estimated time is valid, otherwise the output is declared as invalid.
If a valid solution in inertial frame is computed, then in addition the position and velocity in Earth (E) frame, the Nadir matrix "NJ" and the orbital rate vector in Nadir frame (N) are computed and output.

Syntax:

```
[Data, Status, States, Params]= oop(1)
[Data, Status, States, Params]= oop(1, Params)
[Data, Status, States, Params]= oop(0, Params, GnsrMeasProc, UnitsStatusExpected,
TimeEst, EphemerisDet, States)
```

High level inputs:

Component Name	Oneline Description
GnsrMeasProc	Processed GNSR measurements
UnitsStatusExpected	System expected status of GNSR
TimeEst	Estimated time
EphemerisDet	Ephemeris determination

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
GnsrMeasProc.isValid	Validity flag for item "GnsrMeasProc"	-		defAocsAlgoIsValidIds
GnsrMeasProc.satPos_E	Processed measured satellite position (E)	m	3	
GnsrMeasProc.satVel_E	Processed measured satellite velocity (E)	m/s	3	
TimeEst.isValid	Validity flag for item "TimeEst"	-		defAocsAlgoIsValidIds
TimeEst.time	Estimated time TT	s	1	
EphemerisDet.dcm_JE	Earth rotation matrix	-	3,3	

Name	Oneline Description	Units	Size	Encoding
UnitsStatusExpected.isUsbl	GNSR usability status	-		defTrueFalseIds

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.satPos_J	Estimated satellite inertial position (J)	m	3	
Data.satVel_J	Estimated satellite inertial velocity (J)	m/s	3	
Data.satPos_E	Estimated satellite earth-fixed position (E)	m	3	
Data.satVel_E	Estimated satellite earth-fixed velocity (E)	m/s	3	
Data.dcm_NJ	Nadir to inertial frame attitude matrix	-	3,3	
Data.orbitRate_N	Orbital rate vector in Nadir frame (N)	rad/s	3	
Data.oopPropDur	Duration the OOP is propagating autonomously	s	1	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isUsblAny	Flags usability of least one unit	-		defAocsAlgoIsValidIds
Status.isProp	Flags if OOP is propagating autonomously	-	1	defAocsAlgoIsValidIds
Status.isUsed	Flags if unit is used	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.numGnsrUnits	Number of GNSR units	-	1	
Params.earthAngRate	Earth angular rate	rad/s	1	
Params.earthRad	Earth radius	m	1	
Params.earthGm	Earth gravitational constant	m3/s2	1	
Params.earthJ2	Earth J2 coefficient	-	1	
Params.earthJ3	Earth J3 coefficient	-	1	
Params.earthJ4	Earth J4 coefficient	-	1	

Parameters Default Value:

Parameter Name	Parameter Default Value
numGnsrUnits	1
earthAngRate	7.2921158553e-05
earthRad	6378136.3
earthGm	398600441500000
earthJ2	0.0010826267
earthJ3	-2.5324e-06
earthJ4	-1.6193e-06

States:

Name	Oneline Description	Units	Size	Encoding
States.satPos_J	Last estimated satellite inertial position (J)	m	3	
States.satVel_J	Last estimated satellite inertial velocity (J)	m	3	
States.oopPropTime	Last propagation time	s	1	
States.oopPropUpdateTime	Last update time with GNSR measurements	s	1	

5.2.2.5 relOrbElemsEst

Estimates relative orbit elements based on Lidar and Camera measurement

The estimated relative orbit elements are computed from the Lidar relative position measurements, relative velocity (average rate of change over last measurement interval), attitude estimate and orbit position and velocity estimates.
The output relative orbital elements are the differences in semi-major axis, along-track separation, eccentricity vector x- and y-components and inclination vector x- and y-components. Note that the normalized relative orbit elements are multiplied with the target semi-major axes before output. Thus their unit is meter.
Note, the relation between "Rtn" and T frame is:
X_T = Y_Rtn, Y_T = Z_Rtn, Z_T = X_Rtn => T = [Y Z X], Rtn = [Z X Y]

Syntax:

```
[Data, Status, States, Params]= relOrbElemsEst(1)
[Data, Status, States, Params]= relOrbElemsEst(1, Params)
[Data, Status, States, Params]= relOrbElemsEst(0, Params, LidarMeasProc, StrSatAttEst,
Oop, TimeEst, UnitsStatusExpected, States)
```

High level inputs:

Component Name	Oneline Description
LidarMeasProc	Processed Lidar measurements
StrSatAttEst	Attitude estimate
Oop	Orbit position and velocity estimate
TimeEst	Time estimate
UnitsStatusExpected	System expected status of Unit

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
RelPosEst.isValidCurrent	Flags current validity of relative position	-		defAocsAlgoIsValidIds
RelPosEst.relPos_B	Relative position measurement	m	3	
StrSatAttEst.isValid	Validity flag for item "StrSatAttEst"	-		defAocsAlgoIsValidIds
StrSatAttEst.satAtt_dcm_BJ	Estimated satellite attitude	-	3,3	
Oop.isValid	Validity flag for item "Oop"	-		defAocsAlgoIsValidIds
Oop.satPos_J	Estimated satellite inertial position (J)	m	3	
Oop.satVel_J	Estimated satellite inertial velocity (J)	m/s	3	

Name	Oneline Description	Units	Size	Encoding
TimeEst.isValid	Validity flag for item "TimeEst"	-		defAocsAlgoIsValidIds
TimeEst.time	Estimated time TT	s	1	

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.relOrbElems	Estimated relative orbit elements	m	6	
Data.targetArgOfLat	Target argument of latitude	rad	1	
Data.dcm_BT	Transformation matrix between body (B) and track frame (T)	-	3,3	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.targetArgOfPeriapsis	Target orbit argument of periapsis	rad	1	
Params.targetMeanMotion	Target mean motion	rad/s	1	
Params.targetMjdPeriapsis_TT	Target time of periapsis passage	d	1	
Params.pdiffParams	Pseudo differentiation parameter		5	
Params.filter.A	Filter discrete-time dynamic matrix			
Params.filter.B	Filter discrete-time input matrix			
Params.filter.C	Filter discrete-time output matrix			
Params.filter.D	Filter discrete-time feedthrough matrix			

Parameters Default Value:

Parameter Name	Parameter Default Value
targetArgOfPeriapsis	1.570796326794897
targetMeanMotion	0.001
targetMjdPeriapsis_TT	60000
pdiffParams	[0.9199999999999999, 0.125, -0.1024000000000001, 0.16,

Parameter Name	Parameter Default Value
	1.562499999999999]
filter.signalDim	6
filter.A	0
filter.B	[0, 0, 0, 0, 0, 0]
filter.C	[0; 0; 0; 0; 0; 0]
filter.D	[1, 0, 0, 0, 0, 0; 0, 1, 0, 0, 0, 0; 0, 0, 1, 0, 0, 0; 0, 0, 0, 1, 0, 0; 0, 0, 0, 0, 1, 0; 0, 0, 0, 0, 0, 1]

States:

Name	Oneline Description	Units	Size	Encoding
States.pdiff	Pseudo differentiation state		3	
States.initRelVelComputation	Pseudo differentiation initialisation flag		1	
States.filter.x	Filter state vector			
States.filter.u	Filter input vector			

5.2.2.6 rmuEsSatAttDet

Determines Nadir attitude angles from Earth direction and inertial rate

The Nadir attitude angles Roll-Pitch-Yaw are determined from the ES measured Earth vector and the estimated orbital rate. As input rate the ES Earth vector rate and the RMU rate are used.
Depending on the actual and the specified modes, orbital rate and related yaw angle can be computed. If not possible the attitude will only have valid Roll-Pitch angles.
The additional output orbital rate is the difference of the inertial RMU measured rate and the rate determined from the change of the ES Earth vector which is relative to the nadir frame which is equal to the sum of the orbital rate vector and the satellite rate about the Earth vector.
The other additional output us half cone Earth angle which is defined as the angle of the Earth vector to the z-axis of the body frame.

Syntax:

```
[Data, Status, States, Params]= rmuEsSatAttDet(1)
[Data, Status, States, Params]= rmuEsSatAttDet(1, Params)
[Data, Status, States, Params]= rmuEsSatAttDet(0, Params, RmuSatRateEst, EsEarthDirEst,
aocsMode, States)
```

High level inputs:

Component Name	Oneline Description
RmuSatRateEst	Estimated satellite rate from RMU measurements
EsEarthDirEst	Estimated Earth direction and rate from ES measurements
aocsMode	AOCS mode information

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
RmuSatRateEst.isValid	Validity flag for item "RmuSatRateEst"	-		defAocsAlgoIsValidIds
RmuSatRateEst.satRate_B	Estimated satellite rate in body frame	rad/s	3	
EsEarthDirEst.earthDir_B	Estimated Earth direction in body frame	-	3	
EsEarthDirEst.earthRate_B	Estimated Earth rate in body frame	rad/s	3	
EsEarthDirEst.isValidEarthDir	Validity flag for Earth direction component output	-		defAocsAlgoIsValidIds
EsEarthDirEst.isValidEarthRate	Validity flag for Earth rate component output	-		defAocsAlgoIsValidIds
aocsMode	AOCS mode/submode ID	-	1	

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.isValidAttRp	Validity flag for roll-pitch attitude output	-	1	defAocsAlgoIsValidIds
Data.isValidAttRpy	Validity flag for roll-pitch-yaw attitude output	-	1	defAocsAlgoIsValidIds
Data.isValidEarthAng	Validity flag for Earth angle output	-	1	defAocsAlgoIsValidIds
Data.isValidOrbitRate	Validity flag for orbit rate vector output	-	1	defAocsAlgoIsValidIds
Data.satAtt_rp_BN	Determined satellite attitude roll-pitch	rad	2	
Data.satAtt_rpy_BN	Determined satellite attitude roll-pitch-yaw	rad	3	
Data.earthAng	Earth deviation angle	rad	1	
Data.orbitRate_B	Orbit rate vector in body frame	rad/s	3	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isValidAttRp	Validity flag for roll-pitch attitude output	-	1	defAocsAlgoIsValidIds
Status.isValidAttRpy	Validity flag for roll-pitch-yaw attitude output	-	1	defAocsAlgoIsValidIds
Status.isValidEarthAng	Validity flag for Earth angle output	-	1	defAocsAlgoIsValidIds
Status.isValidOrbitRate	Validity flag for orbit rate vector output	-	1	defAocsAlgoIsValidIds
Status.isRateLow	Flags rate low condition	-	1	defAocsAlgoIsValidIds
Status.isEarthAngLow	Flags Earth angle low condition	-	1	defAocsAlgoIsValidIds
Status.isYawAngLow	Flags yaw angle low condition	-	1	defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.noYawAngleAocsModes	Mode/submode IDs list for not compute yaw angle	-	-1	
Params.rateLowLim	Upper limit for rate low condition check	rad/s	1	
Params.earthAngLowLim	Upper limit for Earth deviation angle low check	rad	1	

Name	Oneline Description	Units	Size	Encoding
Params.yawAngLowLim	Upper limit for yaw angle low check	rad	1	
Params.filterOrbitRate.A	Orbital rate vector filter discrete-time dynamic matrix		1	
Params.filterOrbitRate.B	Orbital rate vector filter discrete-time input matrix		3	
Params.filterOrbitRate.C	Orbital rate vector filter discrete-time output matrix		3	
Params.filterOrbitRate.D	Orbital rate vector filter discrete-time feedthrough matrix		3,3	
Params.filterYawAng.A	Yaw angle filter discrete-time dynamic matrix		1	
Params.filterYawAng.B	Yaw angle filter discrete-time input matrix		1	
Params.filterYawAng.C	Yaw angle filter discrete-time output matrix		1	
Params.filterYawAng.D	Yaw angle filter discrete-time feedthrough matrix		1	

Parameters Default Value:

Parameter Name	Parameter Default Value
noYawAngleAocsModes	[201, 203]
rateLowLim	0.005235987755982988
earthAngLowLim	0.3490658503988659
yawAngLowLim	0.7853981633974483
filterOrbitRate.signalDim	3
filterOrbitRate.A	0
filterOrbitRate.B	[0, 0, 0]
filterOrbitRate.C	[0; 0; 0]
filterOrbitRate.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]
filterYawAng.signalDim	1
filterYawAng.A	0
filterYawAng.B	0
filterYawAng.C	0
filterYawAng.D	1

States:

Name	Oneline Description	Units	Size	Encoding
States.yawAng	Last computed yaw angle	rad	1	
States.filterOrbitRate.x	Orbital rate vector filter state vector		1	
States.filterOrbitRate.u	Orbital rate vector filter input vector		3	
States.filterYawAng.x	Yaw angle filter state vector		1	
States.filterYawAng.u	Yaw angle filter input vector		1	

5.2.2.7 relPosEst

Estimates relative position using Lidar and Cam

Selects Lidar or Camera depending on distance, availability of Lidar/Cam and of relative orbit estimates

- - If no valid relative orbit estimates exist it priorises the first available Lidar
- - If along track distance smaller than a threshold it priorises the first available Lidar
- - If along track distance bigger than a threshold it priorises the first available Camera

In addition to output the relative position measurements of the currently selected unit the last value is maintained as state, to allow CAM also be performed in case Lidar and Cam are not present. Two validity outputs "isValidCurrent" and "isValidState" flag this.

Syntax:

```
[Data, Status, States, Params]= relPosEst(1)
[Data, Status, States, Params]= relPosEst(1, Params)
[Data, Status, States, Params]= relPosEst(0, Params, ...
LidarMeasProc, UnitsStatusExpectedLidar, CamMeasProc, UnitsStatusExpectedCam, ...
RelOrbElemsEst, States)
```

High level inputs:

Component Name	Oneline Description
LidarMeasProc	Processed Lidar measurements
UnitsStatusExpectedLidar	System expected status of Lidar
CamMeasProc	Processed Cam measurements
UnitsStatusExpectedCam	System expected status of Cam struct>
RelOrbElemsEst	Relative orbital elements estimate

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
LidarMeasProc.isValid	Validity flag for item "LidarMeasProc"	-		defAocsAlgoIsValidIds
LidarMeasProc.relPos_B	Processed Lidar relative position measurement (B)	m	3	
UnitsStatusExpectedLidar.isUsbl	Unit usability status	-		defTrueFalseIds
CamMeasProc.isValid	Validity flag for item "CamMeasProc"	-		defAocsAlgoIsValidIds
CamMeasProc.relDir_B	Processed Cam relative direction measurement (B)	-	3	
CamMeasProc.relDist	Processed Cam relative	-	3	

Name	Oneline Description	Units	Size	Encoding
	distance measurement (B)			
UnitsStatusExpectedCam.isUsbl	Unit usability status	-		defTrueFalseIds
RelOrbElemsEst.isValid	Validity flag for item "RelOrbElemsEst"	-		defAocsAlgoIsValidIds
RelOrbElemsEst.relOrbElems	Estimated relative orbit elements	m	6	

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.isValidCurrent	Flags current validity of relative position	-		defAocsAlgoIsValidIds
Data.isValidState	Flags "persistent" validity of relative position	-		defAocsAlgoIsValidIds
Data.relPos_B	Relative position estimate	m	3	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isValidCurrent	Flags current validity of relative position	-		defAocsAlgoIsValidIds
Status.isValidState	Flags "persistent" validity of relative position	-		defAocsAlgoIsValidIds
Status.isUsblAny	Flags usability of least one Lidar or Cam	-		defAocsAlgoIsValidIds
Status.isUsedLidar	Flags if Lidar unit is used	-		defAocsAlgoIsValidIds
Status.isUsedCam	Flags if Cam unit is used	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.numUnitsLidar	Number of Lidar units	-	1	
Params.numUnitsCam	Number of Cam units	-	1	
Params.switchAlongTrackDistLidarCam	Along track distance on which priority changes to Lidar or Cam	m	1	

Parameters Default Value:

Parameter Name	Parameter Default Value
numUnitsLidar	2
numUnitsCam	2
switchAlongTrackDistLidarCam	800

States:

Name	Oneline Description	Units	Size	Encoding
States.relPos_B	Relative position	m	3	
States.isValidState	Flags "persistent" validity of relative position	-		defAocsAlgoIsValidIds

5.2.2.8 rmuSatRateEst

Estimates the RMU satellite rate using a selection method

The RMU rate will be selected according to its validity and usability in the order specified in the selection sequence.

Syntax:

```
[Data, Status, States, Params]= rmuSatRateEst(1)
[Data, Status, States, Params]= rmuSatRateEst(1, Params)
[Data, Status, States, Params]= rmuSatRateEst(0, Params, rmuMeasProc,
UnitsStatusExpected, States)
```

High level inputs:

Component Name	Oneline Description
rmuMeasProc	Processed Rate Measurement Unit measurements
UnitsStatusExpected	System expected status of Unit

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
rmuMeasProc.isValid	Validity flag for item "rmuMeasProc"	-		defAocsAlgoIsValidIds
rmuMeasProc.satRate_B	Processed measured satellite rate	rad/s	3	
UnitsStatusExpected.isUsbl	Unit usability status	-		defTrueFalseIds

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.satRate_B	Estimated satellite rate in body frame	rad/s	3	
Data.selId	ID of selected RMU	-	1	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isUsblAny	Flags usability of least one unit	-		defAocsAlgoIsValidIds
Status.isUsed	Flags if unit is used	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.numUnits	Number of units	-		
Params.selSeq	Selection sequence	-		

Parameters Default Value:

Parameter Name	Parameter Default Value
numUnits	1
selSeq	1

States:

None

5.2.2.9 rwAngMomFricEst

Estimates the angular momentum and the wheel friction

The estimation of the friction torque is made by a Kalman filter whose states are the absolute value of the friction torque and the angular momentum. The angular momentum is propagated thanks to the knowledge of the physical torques applied on the wheels at the previous time step. The friction torque propagation model simply consists in a constant absolute value. The angular momentum innovation is computed using the processed rate measurements, and this innovation is used for the correction of both states. The sign of the friction torque is determined using the angular momentum and the applied torque at the previous time step. The friction torque sign is usually the opposite of the angular momentum sign, but when the angular momentum is too close to zero (zero-crossing zone) to make the identification of its sign possible, the opposite of the sign of the last physical torque applied on the wheels is chosen instead. In zero crossing zones, this torque is normally very high (forced to a maximum value to minimize the zero crossing time), therefore its sign should be easily determined when needed.

Syntax:

```
[Data, Status, States, Params]= rwAngMomFricEst(1)
[Data, Status, States, Params]= rwAngMomFricEst(1, Params)
[Data, Status, States, Params]= rwAngMomFricEst(0, Params, RwMeasProc,
UnitsStatusExpected, RwCmd, rwId, States)
```

High level inputs:

Component Name	Oneline Description
RwMeasProc	Reaction Wheel measurement processing
UnitsStatusExpected	System expected status of Unit
RwCmd	Reaction Wheel commanding (previous)
rwId	ID of the RW, simple loop index

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
RwMeasProc.isValid	Validity flag for item "RwMeasProc"	-		defAocsAlgoIsValidIds
RwMeasProc.rwAngMom	RW measured angular momentum	Nms	1	
UnitsStatusExpected.isUsbl	Unit usability status	-		defTrueFalseIds
RwCmd.isValid	Validity flag for item "RwCmd"	-		defAocsAlgoIsValidIds
RwCmd.trqCmd	RW commanded torque	Nm	numUnits	
rwId	ID of the RW, simple loop index			

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.rwAngMom	RW estimated angular momentum	Nms	1	
Data.rwFricTrq	Estimated friction torque	Nm	1	
Data.rwAngMomInnov	Angular momentum innovation	Nms	1	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.sampleTime	Sample time		1	
Params.rwAngMomUpdGain	Angular momentum update gain	-		
Params.rwFricTrqAbsUpdGain	Friction torque update gain	1/s	1	
Params.rwFricTrqSignDetectLim	Threshold for sign detection	-	1	

Parameters Default Value:

Parameter Name	Parameter Default Value
sampleTime	1
rwAngMomUpdGain	0.1
rwFricTrqAbsUpdGain	0.001
rwFricTrqSignDetectLim	0.001

States:

Name	Oneline Description	Units	Size	Encoding
States.isValid	Validity flag for item "States"	-		defAocsAlgoIsValidIds
States.rwAngMom	RW estimated angular momentum	Nms	1	
States.rwFricTrq	Estimated friction torque	Nm	1	
States.rwFricTrqAbs	Estimated magnitude of friction torque	Nm	1	

5.2.2.10 ssSunDirEst

Estimates the Sun direction based on Sun Sensor

Two methods are offered to estimate the Sun direction: Mean value computation and middle value computation of each component.
This is done based on the validity of each measurement and its usability.
The Sun rate is computed from two consecutive estimated Sun directions.

Syntax:

```
[Data, Status, States, Params]= ssSunDirEst(1)
[Data, Status, States, Params]= ssSunDirEst(1, Params)
[Data, Status, States, Params]= ssSunDirEst(0, Params, SsMeasProc, UnitsStatusExpected,
States)
```

High level inputs:

Component Name	Oneline Description
SsMeasProc	Processed Sun Sensor measurement
UnitsStatusExpected	System expected status of Unit

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
SsMeasProc.isValid	Validity flag for item "SsMeasProc"	-		defAocsAlgoIsValidIds
SsMeasProc.sunDir_B	Measured Sun direction in body frame	-	3	
SsMeasProc.sunRate_B	Computed Sun direction in body frame	rad/s	3	
UnitsStatusExpected.isUsbl	Unit usability status	-		defTrueFalseIds

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.isValidSunDir	Validity flag for Sun direction component output	-		defAocsAlgoIsValidIds
Data.isValidSunRate	Validity flag for Sun rate component output	-		defAocsAlgoIsValidIds
Data.sunDir_B	Estimated Sun direction in body frame	-	3	

Name	Oneline Description	Units	Size	Encoding
Data.sunRate_B	Estimated Sun rate in body frame	rad/s	3	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isValidSunDir	Validity flag for Sun direction component output	-		defAocsAlgoIsValidIds
Status.isValidSunRate	Validity flag for Sun rate component output	-		defAocsAlgoIsValidIds
Status.isUsblAny	Flags usability of least one unit	-		defAocsAlgoIsValidIds
Status.isSunDetected	Flags if Sun is detected (i.e. any valid measurement)	-	1	
Status.isUsed	Flags if unit is used	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.numUnits	Number of units	-		
Params.method	Estimation method (2:mean,3:middle)	-	1	
Params.filterSunDir.A	Sun direction filter discrete-time dynamic matrix		1	
Params.filterSunDir.B	Sun direction filter discrete-time input matrix		3	
Params.filterSunDir.C	Sun direction filter discrete-time output matrix		3	
Params.filterSunDir.D	Sun direction filter discrete-time feedthrough matrix		3,3	
Params.filterSunRate.A	Sun rate filter discrete-time dynamic matrix		1	
Params.filterSunRate.B	Sun rate filter discrete-time input matrix		3	
Params.filterSunRate.C	Sun rate filter discrete-time output matrix		3	
Params.filterSunRate.D	Sun rate filter discrete-time feedthrough matrix		3,3	

Parameters Default Value:

Parameter Name	Parameter Default Value
numUnits	1
method	2
filterSunDir.signalDim	3
filterSunDir.A	0

Parameter Name	Parameter Default Value
filterSunDir.B	[0, 0, 0]
filterSunDir.C	[0; 0; 0]
filterSunDir.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]
filterSunRate.signalDim	3
filterSunRate.A	0
filterSunRate.B	[0, 0, 0]
filterSunRate.C	[0; 0; 0]
filterSunRate.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]

States:

Name	Oneline Description	Units	Size	Encoding
States.filterSunDir.x	Sun direction filter state vector		1	
States.filterSunDir.u	Sun direction filter input vector		3	
States.filterSunRate.x	Sun rate filter state vector		1	
States.filterSunRate.u	Sun rate filter input vector		3	

5.2.2.11 strSatAttEst

Attitude and rate fusion of up to two independent measurements

Attitude and rates are fused independent of each other relying on the measurements for STR attitude and for STR rate.
If only one valid measurement is existing, then the measurement value is taken directly for both, attitude and rate. If no valid measurement is existing for the attitude, then the attitude output is always invalidated. Instead if no valid measurement is existing for the rate, the last rate value is used as output, but only if a previous rate estimation was valid.
Attitude measurements from two STRs are fused. The attitude fusion is based on least-squares and is done in four steps:

- - computation of difference attitude between input attitudes
- - computation of error attitude using weighting matrix
- - correction of input attitude 1 using error attitude from step 2
- - normalization of corrected attitude from step 3

Rate measurements derived from two STRs are fused according to a given rate fusion weighting matrix in a similar way as for the attitude.

Syntax:

```
[Data, Status, States, Params]= strSatAttEst(1)
[Data, Status, States, Params]= strSatAttEst(1, Params)
[Data, Status, States, Params]= strSatAttEst(0, Params, StrMeasProc,
UnitsStatusExpected, States)
```

High level inputs:

Component Name	Oneline Description
StrMeasProc	Processed Star Tracker measurements
UnitsStatusExpected	System expected status of Unit

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
StrMeasProc.isValidAtt	Validity flag for attitude output	-		defAocsAlgoIsValidIds
StrMeasProc.isValidRate	Validity flag for rate output	-		defAocsAlgoIsValidIds
StrMeasProc.satAtt_dcm_BJ	Processed measured satellite attitude	-	3,3	
StrMeasProc.satRate_B	Computed satellite rate (B)	rad/s	3	
UnitsStatusExpected.isUsbl	Unit usability status	-	2	defTrueFalseIds

Outputs:

Name	Oneline Description	Units	Size	Encoding

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.isValidAtt	Validity flag for attitude output	-		defAocsAlgoIsValidIds
Data.isValidRate	Validity flag for rate output	-		defAocsAlgoIsValidIds
Data.satAtt_dcm_BJ	Estimated satellite attitude	-	3,3	
Data.satRate_B	Estimated satellite rate	rad/s	3	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isValidAtt	Validity flag for attitude output	-		defAocsAlgoIsValidIds
Status.isValidRate	Validity flag for rate output	-		defAocsAlgoIsValidIds
Status.isCrossCheckOk	Flags cross check results	-	2	defAocsAlgoIsValidIds
Status.isUsblAny	Flags usability of least one unit	-		defAocsAlgoIsValidIds
Status.isAttFused	Flags that attitudes are fused	-		defAocsAlgoIsValidIds
Status.isRateFused	Flags that rates are fused	-		defAocsAlgoIsValidIds
Status.isRateHold	Flags if rate is held without valid update	-		defAocsAlgoIsValidIds
Status.isUsed	Flags if unit is used	-	2	defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.attFusionWeightMat	Attitude fusion weight matrix	-	3,3	
Params.rateFusionWeightMat_B	Rate fusion weight matrix	-	3,2*3	
Params.crossCheckAngLim	Attitude cross check angle limit	rad	1	

Parameters Default Value:

Parameter Name	Parameter Default Value
attFusionWeightMat	[0.5, 0, 0; 0, 0.5, 0; 0, 0, 0.5]
rateFusionWeightMat_B	[0.5, 0, 0, 0.5, 0, 0; 0, 0.5, 0, 0, 0.5, 0; 0, 0, 0.5, 0, 0, 0.5]
crossCheckAngLim	0.174532925199433

States:

Name	Oneline Description	Units	Size	Encoding
States.satRate_B	Last estimated satellite rate	rad/s	3	
States.isValid	Last validity flag od AocsAlgo component output	-		defAocsAlgoIsValidIds

5.2.2.12 timeEst

Estimates TT derived from GPS OBT for usage in AocsAlgo

Transforms the measured OBT time given as GPS time with GPS reference 8.1.1980 to TT as classical MJD in seconds.
The validity and usability of the measured time is considered.

Syntax:

```
[Data, Status, States, Params]= timeEst(1)
[Data, Status, States, Params]= timeEst(1, Params)
[Data, Status, States, Params]= timeEst(0, Params, ClkMeasProc, UnitsStatusExpected,
States)
```

High level inputs:

Component Name	Oneline Description
ClkMeasProc	Processed measured onboard time OBT
UnitsStatusExpected	System expected status of Unit

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
ClkMeasProc.isValid	Validity flag for item "ClkMeasProc"	-		defAocsAlgoIsValidIds
ClkMeasProc.obtTime	Measured onboard time OBT	s	1	
UnitsStatusExpected.isUsbl	Unit usability status	-	1	defTrueFalseIds

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.time	Estimated time TT	s	1	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isUsblAny	Flags usability of least one unit	-		defAocsAlgoIsValidIds

Parameters:

None

Parameters Default Value:

None

States:

None

5.2.3 Guidance

5.2.3.1 ctrlRefEarth

Computes the Earth reference attitude and rate for control

Reference attitude is computed for the geocentric nadir frame (N). From the change of the reference attitude the reference rate is computed.

Syntax:

```
[Data, Status, States, Params]= ctrlRefEarth(1)
[Data, Status, States, Params]= ctrlRefEarth(1, Params)
[Data, Status, States, Params]= ctrlRefEarth(0, Params, Oop, States)
```

High level inputs:

Component Name	Oneline Description
Oop	Orbit position and velocity estimate

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
Oop.isValid	Validity flag for item "Oop"	-		defAocsAlgoIsValidIds
Oop.satPos_J	Estimated satellite inertial position (J)	m	3	
Oop.satVel_J	Estimated satellite inertial velocity (J)	m/s	3	

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.satAtt_dcm_NJ	Reference attitude from inertial (J) to nadir frame (N)	-	3,3	
Data.satRate_N	Reference rate in nadir frame (N)	rad/s	3	

Name	Oneline Description	Units	Size	Encoding
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.sampleTime	Sample time	s	1	

Parameters Default Value:

Parameter Name	Parameter Default Value
sampleTime	1

States:

Name	Oneline Description	Units	Size	Encoding
States.satAtt_dcm_NJ	Reference attitude from inertial (J) to nadir frame (N)	-	3,3	
States.isInitRateRefComputation	Status of initialization of rate reference computation	-	1	

5.2.4 Controller

5.2.4.1 asmEarthAcqCtrl

Controller for Earth acquisition using RMU, ES, MAG commanding MTQ and RCS

The controller has the purpose to align the z-axis of the spacecraft's control frame with the measured earth direction. This is done using a rate controller which rotates the spacecraft around an axis perpendicular to the plane spanned by the spacecraft's z-axis and the earth vector. The reference acquisition rate is set in three steps which are chosen based on the magnitude of the angle between actual and desired earth direction. If the angle is below the smallest angle threshold the acquisition rate is set to zero. In case the spacecraft is upside down (earth error angle 180 deg) a reference rate around the roll axis is introduced.

Syntax:

```
[Data, Status, States, Params]= asmEarthAcqCtrl(1)
[Data, Status, States, Params]= asmEarthAcqCtrl(1, Params)
[Data, Status, States, Params]= asmEarthAcqCtrl(0, Params, RmuEsSatAttDet,
EsEarthDirEst, RmuSatRateEst, MagFieldEst, States)
```

High level inputs:

Component Name	Oneline Description
RmuEsSatAttDet	Estimated Earth attitude and inertial rate
EsEarthDirEst	Estimated Earth direction
RmuSatRateEst	Estimated satellite rates in body frame
MagFieldEst	Estimated magnetic field

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
RmuEsSatAttDet.earthAng	Earth deviation angle	rad	1	
RmuEsSatAttDet.isValidEarthAng	Validity flag for Earth angle output	-	1	defAocsAlgoIsValidIds
EsEarthDirEst.earthDir_B	Estimated Earth direction in body frame	-	3	
EsEarthDirEst.isValidEarthDir	Validity flag for Earth direction component output	-		defAocsAlgoIsValidIds
RmuSatRateEst.isValid	Validity flag for item "RmuSatRateEst"	-		defAocsAlgoIsValidIds
RmuSatRateEst.satRate_B	Estimated satellite rate in body frame	rad/s	3	
MagFieldEst.isValidMagField	Validity flag for magnetic	-		defAocsAlgoIsValidIds

Name	Oneline Description	Units	Size	Encoding
	field output			
MagFieldEst.magField_B	Estimated magnetic field (body frame)	T	3	

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.mtqTrq_B	Commanded torque for Magnetorquers (B)	Nm	3	
Data.rcsTrq_B	Commanded torque for Reaction Control System (B)	Nm	3	
Data.rateErr_B	Satellite rate error	rad/s	3	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.rateRefLim	Reference acquisition rate (3 - high, 1 - low)	rad/s	3	
Params.earthAngLim	Earth angular limits for acquisition rate (3-high, 1- low)	rad	3	
Params.earthAcqKd_B	Proportional controller gains	Nms/rad	3	
Params.satMagMom_B	Residual magnetic moment of satellite	Am2	3	
Params.mtqTrqLim	MTQ torque limit per axis	Nm	1	
Params.mtqAtt_dcm_BUn	Unit alignment matrix	-	3,3	
Params.mtqMagMomLim_U	MTQ maximum magnetic moment per axis (U)	Am2	3	
Params.rcsTrqLim	RCS deadband torque limit	Nm	1	
Params.filter.A	Filter discrete-time dynamic matrix		1	
Params.filter.B	Filter discrete-time input matrix		3	
Params.filter.C	Filter discrete-time output matrix		3	
Params.filter.D	Filter discrete-time feedthrough matrix		3,3	

Parameters Default Value:

Parameter Name	Parameter Default Value
rateRefLim	[0.0026, 0.0035, 0.0052]
earthAngLim	[0.035, 0.52, 0.87]
earthAcqKd_B	[90; 200; 200]
satMagMom_B	[0; 0; 0]
mtqTrqLim	0.01
mtqAtt_dcm_BUn	[1, 0, 0; 0, 1, 0; 0, 0, 1]
mtqMagMomLim_U	[140; 140; 140]
rcsTrqLim	0
filter.signalDim	3
filter.A	0
filter.B	[0, 0, 0]
filter.C	[0; 0; 0]
filter.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]

States:

Name	Oneline Description	Units	Size	Encoding
States.filter.x	Filter state vector			
States.filter.u	Filter input vector			

5.2.4.2 asmMagRateDampCtrl

Proportional controller for rate damping

This controller gets as input an estimated rate and using parameterizable controller parameters determines torque commands using a simple proportional control feedback in opposite direction.

Syntax:

```
[Data, Status, States, Params]= rateDampCtrl(1)
[Data, Status, States, Params]= rateDampCtrl(1, Params)
[Data, Status, States, Params]= rateDampCtrl(0, Params, RateEst, States)
```

High level inputs:

Component Name	Oneline Description
RateEst	Estimated rate

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
RateEst.isValid	Validity flag for item "RateEst"	-		defAocsAlgoIsValidIds
RateEst.magRate_B	Estimated rate	rad/s	3	

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.frc_B	Commanded force	N	3	
Data.trq_B	Commanded torque	Nm	3	
Data.angMom_B	Commanded angular momentum	Nms	3	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.gainKd	Proportional controller gain	-	3	

Parameters Default Value:

Parameter Name	Parameter Default Value
gainKd	[1; 1; 1]

States:

None

5.2.4.3 asmRateDampCtrl

Controller for rate damping using RMU and MAG for commanding MTQ and RCS

The controller uses proportional rate control to damp the satellite rate. Therefore it uses RMU measured/estimated rates. The resulting control torque is distributed to MTQ and RCS. For high rates RCS is the primary actuator while the MTQ is used to keep the satellite in low rate condition with only infrequent THR usage.
The estimated magnetic field is used to compute MTQ torques, the remaining torques are assigned to the RCS.

Syntax:

```
[Data, Status, States, Params]= asmRateDampCtrl(1)
[Data, Status, States, Params]= asmRateDampCtrl(1, Params)
[Data, Status, States, Params]= asmRateDampCtrl(0, Params, RmuSatRateEst, MagFieldEst,
States)
```

High level inputs:

Component Name	Oneline Description
RmuSatRateEst	Estimated satellite rates
MagFieldEst	Estimated magnetic field

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
RmuSatRateEst.isValid	Validity flag for item "RmuSatRateEst"	-		defAocsAlgoIsValidIds
RmuSatRateEst.satRate_B	Estimated satellite rate in body frame	rad/s	3	
MagFieldEst.isValidMagField	Validity flag for magnetic field output	-		defAocsAlgoIsValidIds
MagFieldEst.magField_B	Estimated magnetic field (body frame)	T	3	

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.mtqTrq_B	Commanded torque for Magnetorquers (B)	Nm	3	
Data.rcsTrq_B	Commanded torque for Reaction Control System (B)	Nm	3	
Data.rateErr_B	Satellite rate error	rad/s	3	

Name	Oneline Description	Units	Size	Encoding
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.rateDampKd_B	Proportional controller gains	Nms/rad	3	
Params.rateRef_B	Reference rate	rad/s	3	
Params.satMagMom_B	Residual magnetic moment of satellite	Am2	3	
Params.mtqTrqLim	MTQ torque limit per axis	Nm	1	
Params.mtqAtt_dcm_BUn	Unit alignment matrix	-	3,3	
Params.mtqMagMomLim_U	MTQ maximum magnetic moment per axis (U)	Am2	3	
Params.rcsRateLim	Minimum rate for RCS usage		1	
Params.filter.A	Control torque filter discrete-time dynamic matrix		1	
Params.filter.B	Control torque filter discrete-time input matrix		3	
Params.filter.C	Control torque filter discrete-time output matrix		3	
Params.filter.D	Control torque filter discrete-time feedthrough matrix		3,3	

Parameters Default Value:

Parameter Name	Parameter Default Value
rateDampKd_B	[100; 200; 200]
rateRef_B	[0; 0; 0]
satMagMom_B	[0; 0; 0]
mtqTrqLim	0.01
mtqAtt_dcm_BUn	[1, 0, 0; 0, 1, 0; 0, 0, 1]
mtqMagMomLim_U	[140; 140; 140]
rcsRateLim	0.003490658503988659
filter.signalDim	3
filter.A	0
filter.B	[0, 0, 0]
filter.C	[0; 0; 0]

Parameter Name	Parameter Default Value
filter.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]

States:

Name	Oneline Description	Units	Size	Encoding
States.filter.x	Control torque filter state vector		1	
States.filter.u	Control torque filter input vector		3	

5.2.4.4 asmSteadyStateCtrl

Controller for steady state Earth pointing using RMU, ES, MAG commanding MTQ and RCS

The steady state controller uses PD control for the roll and pitch axis and pure rate control for the yaw axis to keep the satellite in Earth pointing attitude. The thrusters are only used if angle or rate error is outside a defined deadband.

Syntax:

```
[Data, Status, States, Params]= asmSteadyStateCtrl(1)
[Data, Status, States, Params]= asmSteadyStateCtrl(1, Params)
[Data, Status, States, Params]= asmSteadyStateCtrl(0, Params, RmuEsSatAttDet,
RmuSatRateEst, MagFieldEst, States)
```

High level inputs:

Component Name	Oneline Description
RmuEsSatAttDet	Estimated Earth attitude and inertial rate
RmuSatRateEst	Estimated satellite rates in body frame
MagFieldEst	Estimated magnetic field

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
RmuEsSatAttDet.satAtt_rp_BN	Determined satellite attitude roll-pitch	rad	2	
RmuEsSatAttDet.isValidAttRp	Validity flag for roll-pitch attitude output	-	1	defAocsAlgoIsValidIds
RmuSatRateEst.isValid	Validity flag for item "RmuSatRateEst"	-		defAocsAlgoIsValidIds
RmuSatRateEst.satRate_B	Estimated satellite rate in body frame	rad/s	3	
MagFieldEst.isValidMagField	Validity flag for magnetic field output	-		defAocsAlgoIsValidIds
MagFieldEst.magField_B	Estimated magnetic field (body frame)	T	3	

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.mtqTrq_B	Commanded torque for Magnetorquers	Nm	3	

Name	Oneline Description	Units	Size	Encoding
	(B)			
Data.rcsTrq_B	Commanded torque for Reaction Control System (B)	Nm	3	
Data.rateErr_B	Satellite rate error	rad/s	3	
Data.attErr_rp_BN	Attitude error for roll and pitch	rad	2	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.attRef_rp_BN	Reference roll and pitch angle	rad	2	
Params.attErrLim_rp_BN	Maximum attitude error used for roll-pitch control	rad	2	
Params.steadyStateKp_rp_BN	Gains for roll pitch angle control	Nm/rad	2	
Params.steadyStateKd_B	Gains for rate control	Nms/rad	3	
Params.orbitRate	Orbit rate magnitude	rad/s	1	
Params.rcsAngLim_rp_BN	Thruster deadband angle	rad	2	
Params.rcsAngRateLim_rp_BN	Rate limit for the angle deadband	rad/s	2	
Params.rcsRateLim_B	Thruster deadband rate	rad/s	3	
Params.satMagMom_B	Residual magnetic moment of satellite	Am2	3	
Params.mtqTrqLim	MTQ torque limit per axis	Nm	1	
Params.mtqAtt_dcm_BUn	Unit alignment matrix	-	3,3	
Params.mtqMagMomLim_U	MTQ maximum magnetic moment per axis (U)	Am2	3	
Params.rcsTrqLim	RCS deadband torque limit	Nm	1	
Params.satMoiDecoupMat	Satellite MOI decoupling matrix	-	3,3	
Params.filterAttObs.A	Attitude observation filter discrete-time dynamic matrix			
Params.filterAttObs.B	Attitude observation filter discrete-time input matrix			
Params.filterAttObs.C	Attitude observation filter discrete-time output matrix			
Params.filterAttObs.D	Attitude observation filter discrete-time feedthrough matrix			

Name	Oneline Description	Units	Size	Encoding
Params.filterRateObs.A	Rate observation filter discrete-time dynamic matrix			
Params.filterRateObs.B	Rate observation filter discrete-time input matrix			
Params.filterRateObs.C	Rate observation filter discrete-time output matrix			
Params.filterRateObs.D	Rate observation filter discrete-time feedthrough matrix			
Params.filterAttErr.A	Attitude error filter discrete-time dynamic matrix			
Params.filterAttErr.B	Attitude error filter discrete-time input matrix			
Params.filterAttErr.C	Attitude error filter discrete-time output matrix			
Params.filterAttErr.D	Attitude error filter discrete-time feedthrough matrix			
Params.filterRateErr.A	Rate error filter discrete-time dynamic matrix			
Params.filterRateErr.B	Rate error filter discrete-time input matrix			
Params.filterRateErr.C	Rate error filter discrete-time output matrix			
Params.filterRateErr.D	Rate error filter discrete-time feedthrough matrix			

Parameters Default Value:

Parameter Name	Parameter Default Value
attRef_rp_BN	[0; 0]
attErrLim_rp_BN	[0.5235987755982988; 0.5235987755982988]
steadyStateKp_rp_BN	[0.02; 0.2]
steadyStateKd_B	[12; 67; 79]
orbitRate	0.001136
rcsAngLim_rp_BN	[0; 0]
rcsAngRateLim_rp_BN	[0.0003490658503988659; 0.000174532925199433]
rcsRateLim_B	[0.02; 0.02; 0.02]
satMagMom_B	[0; 0; 0]
mtqTrqLim	0.01

Parameter Name	Parameter Default Value
mtqAtt_dcm_BUn	[1, 0, 0; 0, 1, 0; 0, 0, 1]
mtqMagMomLim_U	[140; 140; 140]
rcsTrqLim	0
satMoiDecoupMat	[1, 0, 0; 0, 1, 0; 0, 0, 1]
filterAttObs.signalDim	2
filterAttObs.A	0
filterAttObs.B	[0, 0]
filterAttObs.C	[0; 0]
filterAttObs.D	[1, 0; 0, 1]
filterRateObs.signalDim	3
filterRateObs.A	0
filterRateObs.B	[0, 0, 0]
filterRateObs.C	[0; 0; 0]
filterRateObs.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]
filterAttErr.signalDim	2
filterAttErr.A	0
filterAttErr.B	[0, 0]
filterAttErr.C	[0; 0]
filterAttErr.D	[1, 0; 0, 1]
filterRateErr.signalDim	3
filterRateErr.A	0
filterRateErr.B	[0, 0, 0]
filterRateErr.C	[0; 0; 0]
filterRateErr.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]

States:

Name	Oneline Description	Units	Size	Encoding
States.filterAttObs.x	Filter state vector		1	
States.filterAttObs.u	Filter input vector		2	
States.filterRateObs.x	Filter state vector		1	
States.filterRateObs.u	Filter input vector		3	
States.filterAttErr.x	Filter state vector		1	

Name	Oneline Description	Units	Size	Encoding
States.filterAttErr.u	Filter input vector		2	
States.filterRateErr.x	Filter state vector		1	
States.filterRateErr.u	Filter input vector		3	

5.2.4.5 asmYawAcqCtrl

Controller for yaw acquisition using RMU, ES, MAG commanding MTQ and RCS

The controller performs yaw acquisition around the yaw axis while maintaining Earth pointing. For earth pointing hold, a reference rate and rotation axis are derived from the desired and current earth direction. For yaw acquisition the sign of the reference rate around the yaw axis is determined by the sign of the estimated yaw angle. To avoid frequent change of the acquisition rate sign due to inaccurate yaw angle estimates the yaw angle sign is integrated applying an upper and lower limit to the integrator state. The limited state of the integrator is then used to determine the current acquisition rate sign.

Syntax:

```
[Data, Status, States, Params]= asmYawAcqCtrl(1)
[Data, Status, States, Params]= asmYawAcqCtrl(1, Params)
[Data, Status, States, Params]= asmYawAcqCtrl(0, Params, RmuEsSatAttDet, EsEarthDirEst,
RmuSatRateEst, MagFieldEst, States)
```

High level inputs:

Component Name	Oneline Description
RmuEsSatAttDet	Estimated Earth attitude and inertial rate
EsEarthDirEst	Estimated Earth direction
RmuSatRateEst	Estimated satellite rates in body frame
MagFieldEst	Estimated magnetic field

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
RmuEsSatAttDet.earthAng	Earth deviation angle	rad	1	
RmuEsSatAttDet.isValidEarthAng	Validity flag for Earth angle output	-	1	defAocsAlgoIsValidIds
RmuEsSatAttDet.satAtt_rpy_BN	Determined satellite attitude roll-pitch-yaw	rad	3	
RmuEsSatAttDet.isValidAttRpy	Validity flag for roll-pitch-yaw attitude output	-	1	defAocsAlgoIsValidIds
EsEarthDirEst.earthDir_B	Estimated Earth direction in body frame	-	3	
EsEarthDirEst.isValidEarthDir	Validity flag for Earth direction component output	-		defAocsAlgoIsValidIds
RmuSatRateEst.isValid	Validity flag for item "RmuSatRateEst"	-		defAocsAlgoIsValidIds
RmuSatRateEst.satRate_B	Estimated satellite rate in	rad/s	3	

Name	Oneline Description	Units	Size	Encoding
	body frame			
MagFieldEst.isValidMagField	Validity flag for magnetic field output	-		defAocsAlgoIsValidIds
MagFieldEst.magField_B	Estimated magnetic field (body frame)	T	3	

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.mtqTrq_B	Commanded torque for Magnetorquers (B)	Nm	3	
Data.rcsTrq_B	Commanded torque for Reaction Control System (B)	Nm	3	
Data.rateErr_B	Satellite rate error	rad/s	3	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.sampleTime	Sample time	s	1	
Params.earthAngLim	Earth angular limit for earth hold	rad	1	
Params.rateRefLim	Lower and upper rate limits for Earth hold reference rate	rad/s	2	
Params.yawRateSignYawAngLim	Yaw angle lower limit for acquisition rate sign determination	rad	1	
Params.yawRateSignIntLim	Upper limit for sign integrator state	-	1	
Params.yawRateRef	Reference yaw acquisition rate	rad/s	1	
Params.yawRateIntLim_B	Limit for integrated rate error state	rad	3	
Params.yawAcqKp	Gain for Earth hold reference rate computation	1/s	1	
Params.yawAcqKd_B	Gain for proportional rate control torque computation	Nms/rad	3	
Params.yawAcqKdi_B	Gain for integrated rate control torque computation	Nm/rad	3	
Params.satMagMom_B	Residual magnetic moment of satellite	Am2	3	
Params.mtqTrqLim	MTQ torque limit per axis	Nm	1	

Name	Oneline Description	Units	Size	Encoding
Params.mtqAtt_dcm_BUn	Unit alignment matrix	-	3,3	
Params.mtqMagMomLim_U	MTQ maximum magnetic moment per axis (U)	Am2	3	
Params.rcsTrqLim	RCS deadband torque limit	Nm	1	
Params.satMoiDecoupMat	Satellite MOI decoupling matrix	-	3,3	
Params.filter.A	Filter discrete-time dynamic matrix		1	
Params.filter.B	Filter discrete-time input matrix		3	
Params.filter.C	Filter discrete-time output matrix		3	
Params.filter.D	Filter discrete-time feedthrough matrix		3,3	

Parameters Default Value:

Parameter Name	Parameter Default Value
sampleTime	1
earthAngLim	0.035
rateRefLim	[0.0012, 0.0024]
yawRateSignYawAngLim	0.5
yawRateSignIntLim	600
yawRateRef	0.00174532925199433
yawRateIntLim_B	[0.1, 1, 0.4]
yawAcqKp	0.003
yawAcqKd_B	[80; 190; 190]
yawAcqKdi_B	[1; 0.4; 1]
satMagMom_B	[0; 0; 0]
mtqTrqLim	0.01
mtqAtt_dcm_BUn	[1, 0, 0; 0, 1, 0; 0, 0, 1]
mtqMagMomLim_U	[140; 140; 140]
rcsTrqLim	0
satMoiDecoupMat	[1, 0, 0; 0, 1, 0; 0, 0, 1]
filter.signalDim	3
filter.A	0
filter.B	[0, 0, 0]
filter.C	[0; 0; 0]

Parameter Name	Parameter Default Value
filter.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]

States:

Name	Oneline Description	Units	Size	Encoding
States.yawRateSignInt	Sign integrator state	-	1	
States.yawRateErrInt_B	Integrated rate error state	rad	3	
States.filter.x	Filter state vector		1	
States.filter.u	Filter input vector		3	

5.2.4.6 camCtrl

Executes Collision Avoidance Maneuver (CAM)

The strategy is quite simple. Using as input the last relative position estimate, the CAM is firing away from the line of sight.

Syntax:

```
[Data, Status, States, Params]= camCtrl(1)
[Data, Status, States, Params]= camCtrl(1, Params)
[Data, Status, States, Params]= camCtrl(0, Params, RelPosEst, States)
```

High level inputs:

Component Name	Oneline Description
RelPosEst	Relative position estimate

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
RelPosEst.isValidState	Validity flag for the state of the relative position	-		defAocsAlgoIsValidIds
RelPosEst.relPos_B	Relative position in body frame (B)	m	3	

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.rcsFrc_B	Commanded force for Reaction Control System (B)	N	3	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.frcLim_B	RCS force limit (B)	N	3	

Parameters Default Value:

Parameter Name	Parameter Default Value

Parameter Name	Parameter Default Value
frcLim_B	[22; 22; 22]

States:

None

5.2.4.7 nomAcqCtrl

Attitude Controller for NOM acquisition submode

The aim of this function is to acquire the desired normal mode attitude after switch-over from another main mode. Because initial attitude and rate conditions at switch-over in conjunction with the disturbance environment might exceed the momentum storage capacity of the reaction wheels, support by RCS can be enabled.
The NOM attitude acquisition is performed via rate control with given acquisition rate about the Euler axis of the attitude error matrix. The acquisition rate is defined in 3 steps: a large rate for large attitude errors, a smaller rate for approaching the steady state deadband and zero within the steady state deadband.
Parallel to the acquisition rate control, angular momentum control is performed for the reaction wheels.

Syntax:

```
[Data, Status, States, Params]= nomAcqCtrl(1)
[Data, Status, States, Params]= nomAcqCtrl(1, Params)
[Data, Status, States, Params]= nomAcqCtrl(0, Params, StrSatAttEst, CtrlRefEarth,
MagFieldEst, RwAngMomCtrl, States)
```

High level inputs:

Component Name	Oneline Description
StrSatAttEst	Startracker attitude estimate
CtrlRefEarth	Earth reference attitude and rate
MagFieldEst	Estimated magnetic field
RwAngMomCtrl	Reaction Wheel angular momentum control

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
StrSatAttEst.isValid	Validity flag for item "StrSatAttEst"	-		defAocsAlgoIsValidIds
StrSatAttEst.satAtt_dcm_BJ	Estimated satellite attitude	-	3,3	
StrSatAttEst.satRate_B	Estimated satellite rate	rad/s	3	
CtrlRefEarth.isValid	Validity flag for item "CtrlRefEarth"	-		defAocsAlgoIsValidIds
CtrlRefEarth.satAtt_dcm_NJ	Reference attitude from inertial (J) to nadir frame (N)	-	3,3	
CtrlRefEarth.satRate_N	Reference rate in nadir frame (N)	rad/s	3	
MagFieldEst.isValidMagField	Validity flag for magnetic field output	-		defAocsAlgoIsValidIds

Name	Oneline Description	Units	Size	Encoding
MagFieldEst.magField_B	Estimated magnetic field (body frame)	T	3	
RwAngMomCtrl.isValid	Validity flag for item "RwAngMomCtrl"	-		defAocsAlgoIsValidIds
RwAngMomCtrl.angMom_B	Angular momentum of all RWs in body frame (B)	Nms	3	
RwAngMomCtrl.desatTrq_B	Desaturation torque for RWs angular momentum (B)	Nm	3	

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.mtqTrq_B	Commanded torque for Magnetorquers (B)	Nm	3	
Data.rcsTrq_B	Commanded torque for Reaction Control System (B)	Nm	3	
Data.rwTrq_B	Commanded torque for Reaction Wheels (b)	Nm	3	
Data.rateErr_B	Satellite rate error	rad/s	3	
Data.attErr_rpy_BN	Satellite attitude Roll-Pitch-Yaw error angles	rad	3	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isSteadyState	Flags submode to be in steady state	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.actSelect	ID for selection of actuator: RW (1) or RCS (2)	-	1	
Params.acqRateRef	Reference rates about Euler axis	rad/s	2	
Params.acqAngLim	Angular limits to select control strategy	rad	2	
Params.acqSteadyStateLim	Angular and rate limits for steady state	rad,rad/s	2	
Params.acqKd	Controller gains for rate error	Nms/rad	3	
Params.satMoiDecoupMat	Satellite MOI decoupling matrix	-	3,3	

Name	Oneline Description	Units	Size	Encoding
Params.satMoi	Satellite moment of inertia	kgm2	3,3	
Params.mtqAtt_dcm_BUn	Unit alignment matrix	-	3,3	
Params.mtqMagMomLim_U	MTQ maximum magnetic moment per axis (U)	Am2	3	
Params.filterRateErr.A	Rate error filter discrete-time dynamic matrix		1	
Params.filterRateErr.B	Rate error filter discrete-time input matrix		3	
Params.filterRateErr.C	Rate error filter discrete-time output matrix		3	
Params.filterRateErr.D	Rate error filter discrete-time feedthrough matrix		3,3	
Params.filterTrqCmd.A	Torque command filter discrete-time dynamic matrix		1	
Params.filterTrqCmd.B	Torque command filter discrete-time input matrix		3	
Params.filterTrqCmd.C	Torque command filter discrete-time output matrix		3	
Params.filterTrqCmd.D	Torque command filter discrete-time feedthrough matrix		3,3	

Parameters Default Value:

Parameter Name	Parameter Default Value
actSelect	1
acqRateRef	[0.00174532925199433, 0.002617993877991494]
acqAngLim	[0.08726646259971647, 0.3490658503988659]
acqSteadyStateLim	[0.08726646259971647, 0.0005235987755982988]
acqKd	[100; 350; 350]
satMoiDecoupMat	[1, 0, 0; 0, 1, 0; 0, 0, 1]
satMoi	[100, 0, 0; 0, 100, 0; 0, 0, 100]
mtqAtt_dcm_BUn	[1, 0, 0; 0, 1, 0; 0, 0, 1]
mtqMagMomLim_U	[140; 140; 140]
filterRateErr.signalDim	3
filterRateErr.A	0
filterRateErr.B	[0, 0, 0]
filterRateErr.C	[0; 0; 0]

Parameter Name	Parameter Default Value
filterRateErr.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]
filterTrqCmd.signalDim	3
filterTrqCmd.A	0
filterTrqCmd.B	[0, 0, 0]
filterTrqCmd.C	[0; 0; 0]
filterTrqCmd.D	[1, 0, 0; 0, 1, 0; 0, 0, 1]

States:

Name	Oneline Description	Units	Size	Encoding
States.filterRateErr.x	Rate error filter state vector		1	
States.filterRateErr.u	Rate error filter input vector		3	
States.filterTrqCmd.x	Torque command filter state vector		1	
States.filterTrqCmd.u	Torque command filter input vector		3	

5.2.4.8 relOrbCtrl

Relative orbit controller for formation keeping

This controller uses relative orbit elements generated from relative position measurements and sequentially performs delta-v manoeuvres to maintain a parametrizable relative orbit.
Note, the relation between "Rtn" and T frame is:
X_T = Y_Rtn, Y_T = Z_Rtn, Z_T = X_Rtn => T = [Y Z X], Rtn = [Z X Y]

Syntax:

```
[Data, Status, States, Params]= relOrbCtrl(1)
[Data, Status, States, Params]= relOrbCtrl(1, Params)
[Data, Status, States, Params]= relOrbCtrl(0, Params, RelOrbElemsEst, TimeEst, aocsMode,
States)
```

High level inputs:

Component Name	Oneline Description
RelOrbElemsEst	Relative orbit element estimate
TimeEst	Time estimator data
aocsMode	AOCS mode information

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
RelOrbElemsEst.isValid	Validity flag for item "RelOrbElemsEst"	-		defAocsAlgoIsValidIds
RelOrbElemsEst.relOrbElems	Estimated relative orbital elements	m	6	
RelOrbElemsEst.targetArgOfLat	Target argument of latitude	rad	1	
RelOrbElemsEst.dcm_BT	Transformation matrix between body (B) and track frame (T)	-	3,3	
TimeEst.isValid	Validity flag for item "TimeEst"	-		defAocsAlgoIsValidIds
TimeEst.time	Estimated time TT	s	1	
aocsMode	AOCS mode/submode ID	-	1	

Outputs:

Name	Oneline Description	Units	Size	Encoding

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.relOrbElemsErr	Relative orbital elements error	m	6	
Data.rcsFrc_B	Commanded force for Reaction Control System (B)	N	3	
Data.frc_T	Commanded force in track frame (T)	N	3	
Data.isNewManPlanned	Flags if new maneuver is planned	-		defYesNoIds
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isSteadyState	Flags CAM control to be in steady state	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.earthRad	Earth radius	m		
Params.earthGm	Earth gravitational constant	m3/s2		
Params.relOrbElemsAocsModes	Assignment of AOCS modes/submodes to reference ROEs	-	N	
Params.relOrbElemsRef	Reference relative orbital elements (ROE)	m	6,N	
Params.relOrbElemsLim	Steady state limits of ROE errors	m	6,1	
Params.targetSemiMajorAxis	Target orbit argument of periapsis	rad	1	
Params.targetInclination	Target orbit inclination	rad	1	
Params.targetMeanMotion	Target orbit mean motion	rad/s	1	
Params.interval	Relative orbit maneuver interval	s	1	
Params.satMass	Satellite mass	kg	1	
Params.frcLim_B	RCS force limit	N	3	

Parameters Default Value:

Parameter Name	Parameter Default Value
earthJ2	0.0010826267
earthRad	6378136.3
relOrbElemsAocsModes	0

Parameter Name	Parameter Default Value
relOrbElemsRef	[0; -1000; 0; -200; 0; 200]
relOrbElemsLim	[0; 0; 0; 0; 0; 0]
targetSemiMajorAxis	7178136.3
targetInclination	1.710422666954443
targetMeanMotion	0.001
interval	6283.185307179586
satMass	1500
frcLim_B	[10; 10; 10]

States:

Name	Oneline Description	Units	Size	Encoding
States.manDeltaV	DeltaV in "Rtn" for current maneuver sequence	m/s	4,3	
States.manArgOfLat	Argument of latitude for current maneuver sequence	rad	4	
States.plannedManStartTime_TT	Start time of current/last maneuver sequence	s	1	
States.plannedManEndTime_TT	End time of current/last maneuver sequence	s	1	

5.2.4.9 rwAngMomCtrl

Reaction wheel angular momentum control

Computes the reaction wheel angular momentum control torque from the actual wheel angular momentum.
The reference angular momentum is computed assuming a nadir reference attitude.

Syntax:

```
[Data, Status, States, Params]= rwAngMomCtrl(1)
[Data, Status, States, Params]= rwAngMomCtrl(1, Params)
[Data, Status, States, Params]= rwAngMomCtrl(0, Params, RwMeasProc, States)
```

High level inputs:

Component Name	Oneline Description
RwMeasProc	Reaction Wheel measurement processing

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
RwMeasProc(numUnits).isValid	Validity flag for item "RwMeasProc"	-		defAocsAlgoIsValidIds
RwMeasProc(numUnits).rwAngMom	RW measured angular momentum	Nms	1	

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.angMom_B	Angular momentum of all RWs in body frame (B)	Nms	3	
Data.desatTrq_B	Desaturation torque for RWs angular momentum (B)	Nm	3	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.numUnits	Number of RW units	-	1	
Params.orbitRate	Orbit rate magnitude	rad/s	1	

Name	Oneline Description	Units	Size	Encoding
Params.satMoi	Satellite moment of inertia	kgm2	3,3	
Params.rwInflMat	RW array influence matrix	-	3,numUnits	
Params.angMomKp	Proportional gain for angular momentum control	1/s	3	

Parameters Default Value:

Parameter Name	Parameter Default Value
numUnits	3
orbitRate	0.001136
satMoi	[100, 0, 0; 0, 100, 0; 0, 0, 100]
rwInflMat	[1, 0, 0; 0, 1, 0; 0, 0, 1]
angMomKp	[0.01; 0.01; 0.01]

States:

None

5.2.5 Actuator Commanding

5.2.5.1 dynCmdDistribution

Distributes commanded dynamics to actuators

Commanded force, torque and angular momentum, typically coming from respective controllers, are first limited before distributed AOCS mode dependent to assigned/intended actuators.

Syntax:

```
[Data, Status, States, Params]= dynCmdDistribution(1)
[Data, Status, States, Params]= dynCmdDistribution(1, Params)
[Data, Status, States, Params]= dynCmdDistribution(0, Params, DynCtrlCmd, aocsMode,
States)
```

High level inputs:

Component Name	Oneline Description
DynCtrlCmd	Commanded dynamics
aocsMode	AOCS mode information

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
DynCtrlCmd.isValid	Validity flag for item "DynCtrlCmd"	-		defAocsAlgoIsValidIds
DynCtrlCmd.frc_B	Commanded force	N	3	
DynCtrlCmd.trq_B	Commanded torque	Nm	3	
DynCtrlCmd.angMom_B	Commanded angular momentum	Nms	3	
aocsMode	AOCS mode/submode ID	-	1	

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.rcsFrc_B	Force command for RCS	N	3	
Data.mtqTrq_B	Torque command for Magnetorquers	Nm	3	
Data.rwTrq_B	Torque command for Reaction Wheels	Nm	3	
Data.rwAngMom_B	Angular momentum command for Reaction Wheels	Nms	3	

Name	Oneline Description	Units	Size	Encoding
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isFrcLim	Flags if commanded input forces are limited	-	1	
Status.isTrqLim	Flags if commanded input torques are limited	-	1	
Status.isAngMomLim	Flags if commanded input angular momentum are limited	-	1	

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.frcLim_B	Force limits per axis (absolute value)	N	3	
Params.trqLim_B	Torque limits per axis (absolute value)	Nm	3	
Params.angMomLim_B	Angular momentum limits per axis (absolute value)	N	3	

Parameters Default Value:

Parameter Name	Parameter Default Value
frcLim_B	[1; 1; 1]
trqLim_B	[1; 1; 1]
angMomLim_B	[1; 1; 1]

States:

None

5.2.5.2 mtqCmd

Magnetorquer commanding

Computation of to be applied magnetic moment of the Magnetorquer to fulfil demanded torque as good as possible.
This will be done based on the available magnetic field.
The z-axis is assumed as the axis about which the magnetic moment will be created.
Scalar and bias are applied to adjust the computed value.

Syntax:

```
[Data, Status, States, Params]= mtqCmd(1)
[Data, Status, States, Params]= mtqCmd(1, Params)
[Data, Status, States, Params]= mtqCmd(0, Params, UnitsStatusExpected, DynCmd,
MagFieldEst, States)
```

High level inputs:

Component Name	Oneline Description
UnitsStatusExpected	System expected status of Unit
DynCmd	Dynamics command
MagFieldEst	Estimated magnetic field

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
UnitsStatusExpected.isUsbl	Unit usability status	-		defTrueFalseIds
DynCmd.isValid	Validity flag for item "DynCmd"	-		defAocsAlgoIsValidIds
DynCmd.mtqTrq_B	Commanded torque for Magnetorquers (B)	Nm	3	
MagFieldEst.isValid	Validity flag for item "MagFieldEst"	-		defAocsAlgoIsValidIds
MagFieldEst.magField_B	Estimated magnetic field	T	3	

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.magMomCmd	Commanded magnetic moment	Am2	1	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds

Name	Oneline Description	Units	Size	Encoding
Status.isUsblAny	Flags usability of least one unit	-		defAocsAlgoIsValidIds
Status.isUsed	Flags if unit is used	-	1	defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.att_dcm_BUn	Unit alignment matrix	-	3,3	
Params.magMomLim	Maximum magnetic moment of the Magnetorquer	Am2	1	
Params.scale_B	Scale correction	-	3	
Params.bias_B	Bias correction	Am2	3	

Parameters Default Value:

Parameter Name	Parameter Default Value
att_dcm_BUn	[1, 0, 0; 0, 1, 0; 0, 0, 1]
magMomLim	100

States:

None

5.2.5.3 rcsCmd

Reaction Control System commanding

Converts commanded forces and torques into RCS frame, applying specified thresholds for minimum and maximum forces and torques on each axis.

Syntax:

```
[Data, Status, States, Params]= rcsCmd(1)
[Data, Status, States, Params]= rcsCmd(1, Params)
[Data, Status, States, Params]= rcsCmd(0, Params, UnitsStatusExpected, DynCmd, States)
```

High level inputs:

Component Name	Oneline Description
UnitsStatusExpected	System expected status of Unit
DynCmd	Dynamics command

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
UnitsStatusExpected.isUsbl	Unit usability status	-		defTrueFalseIds
DynCmd.isValid	Validity flag for item "DynCmd"	-		defAocsAlgoIsValidIds
DynCmd.rcsFrc_B	Commanded force for Reaction Control System (B)	N	3	
DynCmd.rcsTrq_B	Commanded torque for Reaction Control System (B)	Nm	3	

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.frcCmd_U	Commanded force for Reaction Control System (U)	N	3	
Data.trqCmd_U	Commanded torque for Reaction Control System (U)	Nm	3	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Status.isUsblAny	Flags usability of least one unit	-		defAocsAlgoIsValidIds
Status.isUsed	Flags if unit is used	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.att_dcm_BUn	Unit alignment matrix	-	3,3	
Params.frcMinLim_U	Minimum applicable force	N	3	
Params.frcMaxLim_U	Maximum applicable force	N	3	
Params.trqMinLim_U	Minimum applicable torque	Nm	3	
Params.trqMaxLim_U	Maximum applicable torque	Nm	3	

Parameters Default Value:

Parameter Name	Parameter Default Value
att_dcm_BUn	[1, 0, 0; 0, 1, 0; 0, 0, 1]
frcMinLim_U	[0, 0, 0]
frcMaxLim_U	[1, 1, 1]
trqMinLim_U	[0, 0, 0]
trqMaxLim_U	[1, 1, 1]

States:

Name	Oneline Description	Units	Size	Encoding
States				

5.2.5.4 rwCmd

Reaction Wheel commanding

Computes the torques which have to be applied on the individual wheel rotors considering the available RW configuration, 3 of 3 or 3 of 4.
Before output the computed torques are limited to the allowable values.
As input the user has to set the parameter "threeRwInvMat". This is simplified for the user with use of the output of support function "computeThreeRwInvMat", which input is the RW influence matrix (size = {3 x numUnits}).

Syntax:

```
[Data, Status, States, Params]= rwCmd(1)
[Data, Status, States, Params]= rwCmd(1, Params)
[Data, Status, States, Params]= rwCmd(0, Params, UnitsStatusExpected, DynCmd,
RwAngMomFricEst, States)
```

High level inputs:

Component Name	Oneline Description
UnitsStatusExpected	System expected status of Unit
DynCmd	
RwAngMomFricEst	

Low level inputs:

Name	Oneline Description	Units	Size	Encoding
UnitsStatusExpected.isUsbl	Unit usability status	-		defTrueFalseIds
DynCmd.rwTrq_B	Commanded torque for the Reaction Wheels (B)	Nm	3	
RwAngMomFricEst.isValid	Validity flag for item "RwAngMomFricEst"	-		defAocsAlgoIsValidIds
RwAngMomFricEst.rwFricTrq	Estimated friction torque	Nm	1	

Outputs:

Name	Oneline Description	Units	Size	Encoding
Data.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds
Data.trqCmd	Commanded torque for the single Reaction Wheels (U)	Nm	numUnits	
Status.isValid	Validity flag for AocsAlgo component output	-		defAocsAlgoIsValidIds

Name	Oneline Description	Units	Size	Encoding
Status.isUsblAny	Flags usability of least one unit	-		defAocsAlgoIsValidIds
Status.isNumRwAvailOk	Flags if the number of available RWs is okay	-		defAocsAlgoIsValidIds
Status.isUsed	Flags if unit is used	-		defAocsAlgoIsValidIds

Parameters:

Name	Oneline Description	Units	Size	Encoding
Params.numUnits	Number of RW units for the configuration (3 | 4)	-	1	
Params.threeRwInvMat	Three wheel inverse matrices	-	3,3 | 4,3,4	
Params.trqCmdLim	Maximum commanded torque	Nm	1	

Parameters Default Value:

Parameter Name	Parameter Default Value
numUnits	3
threeRwInvMat	[1, 0, 0; 0, 1, 0; 0, 0, 1]
trqCmdLim	1

States:

None