

POLITECNICO DI TORINO



**Dottorato di Ricerca in Ingegneria Aerospaziale
XXVI Ciclo**

**Design and verification of
Guidance, Navigation and Control
systems for space applications**

Tutors

Dr. Sabrina Corpino

Prof. Sergio Chiesa

Eng. Franco Fossati

Candidate

Fabrizio Stesina

Thesis defended on April 1st , 2014

In front of a Board of Examiners composed by:

Prof. Franco Bernelli (Politecnico di Milano)

Prof. Lorenzo Trainelli (Politecnico di Milano)

Prof. Cecilia Surace (Politecnico di Torino)

Fabrizio Stesina. *Design and verification of Guidance Navigation and Control systems for Space applications.*

PhD Thesis Politecnico di Torino

© 2014

Version: February 2nd, 2014

E-mail: fabrizio.stesina@polito.it

*In memoria di chi è partito,
a volte troppo presto,
e di chi non è mai arrivato*

Summary

In the last decades, systems have strongly increased their complexity in terms of number of functions that can be performed and quantity of relationships between functions and hardware as well as interactions of elements and disciplines concurring to the definition of the system. The growing complexity remarks the importance of defining methods and tools that improve the design, verification and validation of the system process: effectiveness and costs reduction without loss of confidence in the final product are the objectives that have to be pursued.

Within the System Engineering context, the modern Model and Simulation based approach seems to be a promising strategy to meet the goals, because it reduces the wasted resources with respect to the traditional methods, saving money and tedious works. Model Based System Engineering (MBSE) starts from the idea that it is possible at any moment to verify, through simulation sessions and according to the phase of the life cycle, the feasibility, the capabilities and the performances of the system. Simulation is used during the engineering process and can be classified from fully numerical (i.e. all the equipment and conditions are reproduced as virtual model) to fully integrated hardware simulation (where the system is represented by real hardware and software modules in their operational environment). Within this range of simulations, a few important stages can be defined: algorithm in the loop (AIL), software in the loop (SIL), controller in the loop (CIL), hardware in the loop (HIL), and hybrid configurations among those.

The research activity, in which this thesis is inserted, aims at defining and validating an iterative methodology (based on Model and Simulation approach) in support of engineering teams and devoted to improve the effectiveness of the design and verification of a space system with particular interest in Guidance Navigation and Control (GNC) subsystem. The choice of focusing on GNC derives from the common interest and background of the groups involved in this research program (ASSET at Politecnico di Torino and AvioSpace, an EADS company). Moreover, GNC system is sufficiently complex (demanding both specialist knowledge and system engineer skills) and vital for whatever spacecraft and, last but not least the verification of its behavior is difficult on ground because strong limitations on dynamics and environment reproduction arise.

Considering that the verification should be performed along the entire product life cycle, a tool and a facility, a simulator, independent from the complexity level of the test and the stage of the project, is needed. This thesis deals with the design of the simulator, called *StarSim*, which is the real heart of the proposed methodology. It has been entirely designed and developed from the requirements definition to the software implementation and hardware construction, up to the assembly, integration and verification of the first simulator release. In addition, the development of this technology meet the modern standards on software development and project management. *StarSim* is a unique and self-contained platform: this feature allows to mitigate the risk of incompatibility, misunderstandings and loss of information that may arise using different software, simulation tools and facilities along the various phases. Modularity, flexibility, speed, connectivity, real time operation, fidelity with real world, ease of data management, effectiveness and congruence of the outputs with respect to the inputs are the sought-after features in the *StarSim* design. For every iteration of the methodology, *StarSim* guarantees the possibility to verify the behavior of the system under test thanks to the permanent availability of virtual models, that substitute all those elements not yet available and all the non-reproducible dynamics and environmental conditions. *StarSim* provides a furnished and user friendly database of models and interfaces that covers different levels of detail and fidelity, and supports the updating of the database allowing the user to create customized models (following few,

simple rules). Progressively, pieces of the on board software and hardware can be introduced without stopping the process of design and verification, avoiding delays and loss of resources.

StarSim has been used for the first time with the CubeSats belonging to the e-st@r program. It is an educational project carried out by students and researchers of the “CubeSat Team Polito” in which StarSim has been mainly used for the payload development, an Active Attitude Determination and Control System, but StarSim’s capabilities have also been updated to evaluate functionalities, operations and performances of the entire satellite. AIL, SIL, CIL, HIL simulations have been performed along all the phases of the project, successfully verifying a great number of functional and operational requirements. In particular, attitude determination algorithms, control laws, modes of operation have been selected and verified; software has been developed step by step and the bugs-free executable files have been loaded on the micro-controller. All the interfaces and protocols as well as data and commands handling have been verified. Actuators, logic and electrical circuits have been designed, built and tested and sensors calibration has been performed. Problems such as real time and synchronization have been solved and a complete hardware in the loop simulation test campaign both for A-ADCS standalone and for the entire satellite has been performed, verifying the satisfaction of a great number of CubeSat functional and operational requirements.

The case study represents the first validation of the methodology with the first release of StarSim. It has been proven that the methodology is effective in demonstrating that improving the design and verification activities is a key point to increase the confidence level in the success of a space mission.

Ringraziamenti

L'idea di iniziare il dottorato di ricerca a 30 anni e tornare allo status di studente è stata una delle idee più pazze della mia vita. Ma un'idea pazza che trova un suo compimento resta pazza ma diventa anche un'ottima idea.

Non sarei mai riuscito a completare questa, così come altre sfide della vita, senza l'aiuto di alcune persone fondamentali, che sento di dover ringraziare.

Ringrazio mia mamma Claudia e mio papà Adriano che sono stati sempre i miei primi sostenitori e che hanno contribuito, attraverso il loro supporto e il loro amore per me, a realizzare qualsiasi mio sogno e desiderio. A loro va tutta la mia gratitudine ed il mio amore.

Ringrazio mio fratello Gianluca da sempre fonte di ispirazione per me e massimo esempio di quel giusto mix di intelligenza e volontà che ti consente di arrivare a tagliare qualsiasi traguardo. Ringrazio mia cognata Stefania e soprattutto i miei nipotini Rebecca ed Edoardo: il loro sorriso mi ha sempre dato tanta forza per superare anche i momenti più difficili e tristi.

Ringrazio Sabrina, mia relatrice in questa tesi, ma soprattutto una grande professionista che mi ha insegnato tanto del mestiere di ingegnere e non solo e, ancor di più, una grande amica che mi ha sostenuto oltre che sopportato in tutti questi anni. A lei va un ringraziamento davvero speciale.

Ringrazio tutti coloro che hanno fatto o fanno parte del gruppo di ricerca del Politecnico di Torino in cui ho il privilegio di lavorare. Ringrazio Nicole, amica e sostenitrice fin dai primi giorni (oramai dieci anni fa) della mia collaborazione, i membri del "gruppo storico", Valerio, Alessandro e Guido, e tutti i ragazzi del CubeSat Team, in particolare Raffaele, Lorenzo, Fabio e Gerard. Tutti loro mi hanno insegnato qualcosa e mi hanno arricchito come professionista e ancor più come persona.

Ringrazio il mio amico di sempre Alessio con il quale ho condiviso molte delle sfide e delle gioie che hanno percorso la mia vita e la mia carriera. Grazie per le tante chiacchierate e gli sfoghi davanti ad un paio di buoni *cuba*.

Infine, non posso non ringraziare i miei detrattori e i miei nemici: dimostrare ogni volta che avete torto è sempre un grande piacere e un grande stimolo per me!

Contents

Summary	i
Ringraziamenti.....	iii
List of Tables	xiii
Definitions	xvii
Introduction	1
Chapter 1. Model and Simulation Based approach in System Engineering.....	5
1.1 System Engineering & Programs Management methodologies	5
1.1.1 The space product life cycle phases.....	7
1.1.2 V-model	11
1.2 Focus on verification and validation.....	12
1.2.1 Verification and validation in the space product life cycle	14
1.2.2 Verification strategies.....	17
1.3 Model and Simulation based approach.....	19
1.3.1 M&S based approach: advantages and disadvantages.....	20
1.3.2 Computers importance in the M&S based approach	21
1.3.3 Use of the simulation along the product life cycle	22
1.3.4 Methods, technique and tools	27
1.4 Simulation strategies and configurations	37
1.4.1 Analysis and design simulators	40
1.4.2 Simulators for qualification and acceptance (AIV simulator).....	47
1.4.3 Simulators for qualification, test and operations of ground segment.....	47
1.4.4 In The Loop Configuration.....	49
1.5 Models and the modelling process.....	53
1.5.1 Model architecture.....	53
1.5.2 Level of detail and degree of fidelity of a model.....	54
1.5.3 Type of models	56
Chapter 1 reference.....	61
Chapter 2. GNC system design	64
2.1 What are G(Guidance), N (Navigation) and C (Control)?.....	64
2.2 GNC interactions with other subsystems.....	65
2.3 GNC design: functions, architectures and features	66
2.3.1 GNC control modes	66
2.3.2 GNC functions.....	68
2.3.3 GNC architecture.....	69
2.3.4 GNC in the product life cycle.....	71
2.4 Verification and validation of a GNC system.....	77
2.4.1 Mission definition and feasibility phase	77
2.4.2 Design phase.....	78
2.4.3 Development phase	79
2.4.4 Qualification Phase	81
2.4.5 Verification at system level	82
2.4.6 GNC-ground interface verification	82
2.4.7 On-orbit verification.....	82

2.5	Guidance strategies, Navigation algorithms and Control methods.....	82
2.5.1	Attitude Determination	82
2.5.2	Orbit determination.....	92
2.5.3	Guidance strategies.....	94
2.5.4	Control: methods and techniques	96
2.6	GNC performance.....	100
2.6.1	Performance indicators.....	101
Chapter 2 reference		109
Chapter 3. The simulator.....		111
3.1	The methodology	111
3.2	StarSim simulator.....	115
3.2.1	Why build a <i>in-house</i> simulator?.....	116
3.2.2	The simulator functions	118
3.2.3	Simulator requirements.....	123
3.3	Simulator architecture	140
3.3.1	The Simulation Unit	140
3.3.2	Ground Support Equipment	163
3.3.3	Control Console.....	168
3.3.4	The test object.....	169
3.4	How to use the simulator.....	169
3.4.1	Simulator main window.....	169
3.4.2	Simulator architecture configuration.....	170
3.4.3	Define the models flow	172
3.4.4	Simulation Setup	173
3.4.5	Define the output files.....	174
3.4.6	Simulation execution.....	174
3.4.7	Results evaluation.....	174
Reference Chapter 3		176
Chapter 4. The test case: e-st@r CubeSat.....		177
4.1	CubeSats.....	177
4.1.1	CubeSat verification process and HIL simulation.....	178
4.2	E-st@r Program overview	179
4.2.1	Mission objectives	180
4.2.2	Mission architecture	182
4.2.3	Mission phases.....	184
4.2.4	Modes of operation	186
4.3	CubeSat design.....	191
4.4	A-ADCS design and verification	193
4.4.1	Design of the A-ADCS of e-st@r CubeSats.....	193
4.4.2	M&S based design and verification of the A-ADCS of e-st@r CubeSat.....	197
4.4.3	Test case: lessons learned	267
Reference Chapter 4		268
Chapter 5. Conclusion.....		270
Appendix A: Reference Frames.....		272
Appendix B:Attitude representation		275

Appendix C: In orbit disturbances that affect the GNC operations	278
Appendix D: control techniques.....	279
Appendix E: RD129- ARM9 processor	286
Appendix F: StarSim Core features.....	288
Riassunto.....	289
Curriculum Vitae	291

List of Figures

FIGURE 1: SE SPIRAL APPROACH (BOHEM 1988).....	6
FIGURE 2: PRODUCT LIFE CYCLE	7
FIGURE 3: TRADITIONAL V-MODEL IN SE.....	12
FIGURE 4: V MODEL USING MODEL AND SIMULATION BASED APPROACH FOR THE VERIFICATION [9].....	14
FIGURE 5: VERIFICATION AND VALIDATION PROCESS IN THE SPACE PRODUCT LIFE CYCLE	16
FIGURE 6: SYSTEM DEVELOPMENT BASED ON SIMULATION	24
FIGURE 7: USE OF THE SIMULATION IN THE PRODUCT LIFE CYCLE [7].....	24
FIGURE 8: EXAMPLE OF THE CODE GENERATION MADE BY TASTE	29
FIGURE 9: TYPICAL SW SIMULATION SYSTEM ARCHITECTURE BASED ON SIMWARE	30
FIGURE 10: LOGICAL ARCHITECTURE OF SVF INFRASTRUCTURE AND MODELS.....	31
FIGURE 11: EPOS 2.0 FACILITY	33
FIGURE 12: MATLAB-SIMULINK [®] EXAMPLE	34
FIGURE 13: MODEL BASED DEVELOPMENT AND VERIFICATION ENVIRONMENT (MDVE) - ASTRIUM.....	35
FIGURE 14: ASTRIUM TEST BENCH [28]	36
FIGURE 15: TASKS SUPPORTED BY M&S APPROACH [8].....	37
FIGURE 16: SIMULATION FACILITY ARCHITECTURE COMPONENTS.....	39
FIGURE 17: SCS - SYSTEM CONCEPT SIMULATOR	41
FIGURE 18: FES - FUNCTIONAL ENGINEERING SIMULATOR.....	42
FIGURE 19: FVTB - FUNCTIONAL VALIDATION TEST-BENCH.....	44
FIGURE 20: SVF - SOFTWARE CONFIGURATION.....	45
FIGURE 21: SVF- HARDWARE IN THE LOOP CONFIGURATION.....	46
FIGURE 22: ALGORITHMS IN THE LOOP CONFIGURATION [28].....	50
FIGURE 23: SOFTWARE IN THE LOOP [28]	51
FIGURE 24: CONTROLLER IN THE LOOP [28].....	51
FIGURE 25: HARDWARE IN THE LOOP [28].....	52
FIGURE 26: MODEL ARCHITECTURE.....	53
FIGURE 27: GNC INTERACTIONS WITH MISSION, SYSTEM AND OTHER SUBSYSTEMS ELEMENTS	65
FIGURE 28: BASIC CONTROL FEEDBACK STRUCTURE.....	69
FIGURE 29: DETAILED FEEDBACK CONTROL SYSTEM [4]	70
FIGURE 30: CONTROL ENGINEERING PROCESS.....	72
FIGURE 31: CLOSED LOOP GNC SIMULATION CONFIGURATION DURING THE DESIGN PHASE	79
FIGURE 32: CLOSED LOOP GNC SIMULATION CONFIGURATION AT DEVELOPMENT PHASE	80
FIGURE 33: OPEN LOOP GNC STIMULATION FOR SENSOR HW VERIFICATION.....	81
FIGURE 34: CLOSED LOOP GNC SIMULATION CONFIGURATION AT VERIFICATION STAGE WITH REAL HW	82
FIGURE 35: SCHEMATIC OPERATION OF A KALMAN FILTER.....	91
FIGURE 36: ADDITIVE AND MULTIPLICATIVE UNSTRUCTURED UNCERTAINTIES.....	105
FIGURE 37: STRUCTURED UNCERTAINTIES.....	106
FIGURE 38: SYSTEM WITH A STRUCTURED UNCERTAINTY	107
FIGURE 39: PERFORMANCE ROBUSTNESS ANALYSIS USING SSV	108
FIGURE 40: FLOW CHART OF THE METHODOLOGY	114
FIGURE 41: TOP LEVEL SIMULATOR FUNCTIONS	118
FIGURE 42: FT – SUBFUNCTIONS (I).....	119
FIGURE 43: FT – SUBFUNCTIONS (II).....	119
FIGURE 44: FT – SUBFUNCTIONS (VI).....	119
FIGURE 45: FT – SUBFUNCTIONS (III).....	120
FIGURE 46: FT – SUBFUNCTIONS (IV).....	120
FIGURE 47: FT – SUBFUNCTIONS (V).....	120

FIGURE 48: FT – SUBFUNCTIONS (VII)	120
FIGURE 49: FT – SUBFUNCTIONS (VIII)	121
FIGURE 50: FT – SUBFUNCTIONS (IX)	122
FIGURE 51: GENERAL SIMULATOR'S ARCHITECTURE	140
FIGURE 52: PROCESSES AND THREADS	142
FIGURE 53: WINDOWS OF THE SUCCESSFUL INSTALLATION OF LINUX RT-PATCHED KERNEL.....	148
FIGURE 54: THE IMPORTANCE OF RTOS.....	149
FIGURE 55: SIMULATOR SCHEDULER FLOW CHART	150
FIGURE 56: EXAMPLE OF SETTHREAD.TXT FILE	151
FIGURE 57: CODE SNIPPET ON PROCESS CREATION.....	151
FIGURE 58: SIMULATOR AND SIMULATION SETUP PROCESS.....	152
FIGURE 59: ITERATION TO PARSE THE DATABASE.....	153
FIGURE 60: EXAMPLE OF MODEL SETTING PARAMETERS.....	154
FIGURE 61: IMU SETTINGS WINDOW	155
FIGURE 62: PLOT OF THE OUTPUTS FOR IMU SPARKFUN CHARACTERIZATION	165
FIGURE 63: MOBILE GROUND CONTROL STATION SCHEME	166
FIGURE 64: LAFAYETTE SG7000 ANTENNA.....	167
FIGURE 65: KENWOOD TH-F7E.....	167
FIGURE 66: TNC-7 MULTI.....	167
FIGURE 67: BLOCKS SCHEME OF THE POWER FRONTEND FOR STARSIM	168
FIGURE 68: BLOCKS SCHEME OF THE TM/TC FRONT-END FOR STARSIM.....	168
FIGURE 69: STARSIM MAIN WINDOW	170
FIGURE 70: INITIAL PROCESS SECTION WINDOW.....	170
FIGURE 71: PROCESS SELECTION, MAIN WINDOW	171
FIGURE 72: MODELS SELECTION WINDOW	172
FIGURE 73: SIMULATION PARAMETERS WINDOW.....	173
FIGURE 74: OUTPUTS FILES DEFINITION WINDOW, SENSIBLE VARIABLES SELECTION	174
FIGURE 75: EXAMPLE OF A PLOT MADE AFTER A SIMULATION SESSION WITH GNUPLOT	175
FIGURE 76: E-ST@R PROGRAM GUIDELINES.....	180
FIGURE 77: E-ST@R MISSION OBJECTIVES	182
FIGURE 78: MISSION ARCHITECTURE (E-ST@R-II MISSION).....	183
FIGURE 79: MAIN GCS LOCATION (LEFT), BUILDING AND ANTENNAS (RIGHT).....	184
FIGURE 80: MISSION PHASES AND OPERATIVE MODES	188
FIGURE 81: SWITCH CONFIGURATION IN THE EPS BOARD. CREDIT CLYDESPACE LTD.....	190
FIGURE 82: E-ST@R BLOCKS SCHEME	192
FIGURE 83: E-ST@R-I FLIGHT UNIT.....	192
FIGURE 84: FUNCTIONAL TREE (TOP LEVEL) OF E-ST@R A-ADCS.....	193
FIGURE 85: FUNCTIONAL TREE (PART 1) OF E-ST@R A-ADCS	193
FIGURE 86: FUNCTIONAL TREE (PART 2) OF E-ST@R A-ADCS	194
FIGURE 87: FUNCTIONAL TREE (PART 3) OF E-ST@R A-ADCS	194
FIGURE 88: FUNCTIONAL TREE (PART 4) OF E-ST@R A-ADCS	194
FIGURE 89: A-ADCS INTERFACES WITH THE OTHER SUBSYSTEMS	195
FIGURE 90: BLOCKS SCHEME OF A-ADCS	197
FIGURE 91: E-ST@R-II ADCS OPERATIVE MODES	200
FIGURE 92: AIL SIMULATIONS – PROCESSES SETTINGS.....	201
FIGURE 93: BASIC AIL SIMULATION – MODELS FLOW.....	201
FIGURE 94: BASIC AIL SIMULATION - ORBIT PROPAGATION	202
FIGURE 95: BASIC AIL SIMULATION - EMF GENERATED FROM THE MODEL	202
FIGURE 96: BASIC AIL SIMULATION - DISTURBANCE TORQUES TRENDS.....	203

FIGURE 97: BASIC AIL SIMULATION –DETAIL OF THE ANGULAR VELOCITIES WHEN NO CONTROL TORQUES ARE APPLIED	204
FIGURE 98: BASIC AIL SIMULATION -CONTROL TORQUES TREND AND SIZING	204
FIGURE 99: CONTROL TECHNIQUES AIL SIMULATION – MODELS FLOW: ACTUATORS AND CONTROL LAWS	205
FIGURE 100: CONTROL TECHNIQUES AIL SIMULATION DETUMBLING PHASE USING BDOT	206
FIGURE 101: CONTROL TECHNIQUES AIL SIMULATION DETUMBLING PHASE USING BDOT (DETAIL).....	206
FIGURE 102: CONTROL TECHNIQUES AIL SIMULATION - DETUMBLING PHASE USING WXB CONTROL LAW	206
FIGURE 103: CONTROL TECHNIQUES AIL SIMULATION-DETUMBLING PHASE WITH WXB CONTROL (DETAIL)...	206
FIGURE 104: CONTROL TECHNIQUES AIL SIMULATION- STABILIZATION CONTROL, PID	207
FIGURE 105: CONTROL TECHNIQUES AIL SIMULATION - STABILIZATION WITH LQR.....	207
FIGURE 106: FREE MOTION SEVEN DAYS SIMULATION AND SUCCESSIVE DETUMBLING	208
FIGURE 107: DETERMINATION ALGORITHMS COMPARISON VIA AIL SIMULATION – MODELS FLOW	209
FIGURE 108: DETERMINATION ALGORITHMS COMPARISON VIA AIL SIMULATION –MODELED AND MEASURED EMF	210
FIGURE 109: DETERMINATION ALGORITHMS COMPARISON VIA AIL SIMULATION – SIMULATED AND MEASURED ANGULAR VELOCITIES (DETAIL – FIRST 5000 SECONDS)	211
FIGURE 110: COMPLETE AIL SIMULATION – MODELS FLOW	212
FIGURE 111: COMPLETE AIL SIMULATION: C++ CODE SKELETON	213
FIGURE 112: COMPLETE AIL SIMULATION – C++ CODE SKELETON FOR THE SIMULATION PROCESS IN COMPLETE AIL SIMULATIONS (II).....	214
FIGURE 113: COMPLETE AIL SIMULATION - SATELLITE ATTITUDE (EULER ANGLES).....	215
FIGURE 114: COMPLETE AIL SIMULATION - ESTIMATED QUATERNION USING THE MAGNETOMETER MEASUREMENT (Q-METHOD) AND THE GYROSCOPE MEASUREMENT (EKF).....	215
FIGURE 115: COMPLETE AIL SIMULATION - DIPOLE MOMENT (M), GENERATING CONTROL TORQUE, TREND .	216
FIGURE 116: MAIN ADCS SOFTWARE FLOW CHART	219
FIGURE 117: SIL SIMULATION FOR IMU VERIFICATION: PROCESSES SETTINGS	220
FIGURE 118: SIL SIMULATION FOR IMU VERICATION – “MODELS FLOW” FOR ACQUISITION AND DATA MANAGEMENT.....	221
FIGURE 119: SIL SIMULATION FOR ADCS/OBC PROTOCOLS VERIFICATION – PROCESSES SETTINGS	222
FIGURE 120: SIL SIMULATION FOR ADCS/OBC PROTOCOLS VERIFICATION - “MODELS FLOW”.....	223
FIGURE 121: COMPLETE SIL SIMULATION - PROCESSES SETTINGS.....	224
FIGURE 122: COMPLETE SIL SIMULATIONS – MODELS FLOW	225
FIGURE 123: CIL SIMULATIONS FOR PWM VERIFICATION – PROCESSES SETTINGS	227
FIGURE 124: CIL SIMULATIONS FOR PWM VERIFICATION - MODELS FLOW	228
FIGURE 125: CIL SIMULATIONS FOR PWM VERIFICATION - DUTY CYCLE 99.9% (OSCILLOSCOPE DISPLAY)	229
FIGURE 126: CIL SIMULATIONS FOR PWM VERIFICATION - DUTY CYCLE 0.1% (OSCILLOSCOPE DISPALY)	229
FIGURE 127: CIL SIMULATIONS FOR PWM VERIFICATION	229
FIGURE 128: CIL SIMULATIONS FOR IMU DATA MANAGEMENT - PROCESSES SETTINGS	230
FIGURE 129: CIL SIMULATIONS FOR IMU DATA MANAGEMENT - MODELS FLOW	231
FIGURE 130: CIL SIMULATION FOR OBC/ADCS COMMUNICATION VERIFICATION - PROCESSES AND SETTINGS	232
FIGURE 131: CIL SIMULATION FOR OBC/ADCS COMMUNICATION VERIFICATION - MODELS FLOW	233
FIGURE 132: CIL SIMULATION FOR OBC/ADCS COMMUNICATION VERIFICATION – TEST ARRANGEMENT USING EM	233
FIGURE 133: CIL SIMULATIONS FOR OBC/ADCS COMMUNICATION VERIFICATION - <i>FLAG READ</i> OBC.....	234
FIGURE 134: CIL SIMULATIONS FOR OBC/ADCS COMMUNICATION VERIFICATION - NUMBER OF BITS WRITTEN BY THE ADCS SOFTWARE ON SERIAL-OBC PORT	234
FIGURE 135: SCHEMATIC OF THE ADCS BOARD (I)	235
FIGURE 136: SCHEMATIC OF THE ADCS BOARD (II)	236
FIGURE 137: ADCS BOARD, VIEW FROM EAGLECAD AND FM DURING BASIC ELECTRICAL TESTS.....	236
FIGURE 138: MAGNETIC TORQUERS (MT)	237

FIGURE 139: HIL SIMULATIONS FOR IMU FUNCTIONAL TEST: PROCESSES SETTINGS.....	238
FIGURE 140: HIL SIMULATIONS FOR IMU FUNCTIONAL TEST -MODELS FLOW.....	238
FIGURE 141: HIL SIMULATIONS FOR IMU FUNCTIONAL TEST – ARRANGEMENT OF THE TEST USING FM UNIT	239
FIGURE 142: HIL SIMULATIONS FOR IMU FUNCTIONAL TEST - C++ CODE SKELETON.....	240
FIGURE 143: HIL SIMULATIONS FOR IMU CALIBRATIONS - PROCESSES SETTINGS.....	241
FIGURE 144: HIL SIMULATIONS FOR IMU CALIBRATION - MODELS FLOW.....	242
FIGURE 145: HIL SIMULATION FOR IMU CALIBRATION – ARRANGEMENT OF THE TEST WITH QM.....	243
FIGURE 146: HIL SIMULATION FOR IMU CALIBRATION - X-AXIS GYROSCOPE CHARACTERIZATION	243
FIGURE 147: HIL SIMULATION FOR IMU CALIBRATION - Y-AXIS GYROSCOPE CHARACTERIZATION	243
FIGURE 148: HIL SIMULATION FOR IMU CALIBRATION – Z-AXIS GYROSCOPE CHARACTERIZATION.....	244
FIGURE 149: HIL SIMULATIONS FOR PWM AND CURRENT SENSORS CALIBRATION – PROCESSES SETTINGS	245
FIGURE 150: HIL SIMULATIONS FOR PWM AND CURRENT SENSORS CALIBRATION - MODELS FLOW.....	246
FIGURE 151: HIL SIMULATIONS FOR PWM AND CURRENT SENSORS CALIBRATION . C++ CODE SKELETON.....	247
FIGURE 152: HIL SIMULATIONS FOR PWM AND CURRENT SENSORS CALIBRATION – PICTURE OF THE ARRANGEMENT FOR THE TEST WITH FM	247
FIGURE 153: HIL SIMULATION FOR PWM AND CURRENT SENSORS CALIBRATION - PWM LOGIC CIRCUIT CALIBRATION	248
FIGURE 154: HIL SIMULATION FOR PWM AND CURRENT SENSORS CALIBRATIONS - PWM TUNING	248
FIGURE 155: HIL SIMULATION FOR PWM AND CURRENT SENSORS CALIBRATION.....	248
FIGURE 156: HIL SIMULATIONS FOR COMPLETE A-ADCS VERIFICATION – PROCESSES SETTINGS	249
FIGURE 157: HIL SIMULATION FOR COMPLETE A-ADCS VERIFICATION - "MODELS FLOW".....	251
FIGURE 158: HIL SIMULATION FOR COMPLETE A-ADCS VERIFICATION: C++ CODE GENERATED SKELETON (I)..	252
FIGURE 159: HIL SIMULATION FOR COMPLETE A-ADCS VERIFICATION: C++ CODE GENERATED SKELETON (II).	253
FIGURE 160: HIL SIMULATION FOR COMPLETE A-ADCS VERIFICATION - ARRANGEMENT FOR TEST WITH QM.	253
FIGURE 161: HIL SIMULATION FOR COMPLETE A-ADCS VERIFICATION - BODY ANGULAR VELOCITY.....	254
FIGURE 162: HIL SIMULATION FOR COMPLETE A-ADCS VERIFICATION – ATTITUDE	254
FIGURE 163: BODY ANGULAR VELOCITY WRT ORBITAL FRAME (AXIS X), AIL(BLUE) VS. HIL(RED).....	255
FIGURE 164: BODY ANGULAR VELOCITY WRT ORBITAL FRAME (AXIS Y), AIL(BLUE) VS. HIL(RED).....	255
FIGURE 165: BODY ANGULAR VELOCITY WRT ORBITAL FRAME (AXIS Z), AIL(BLUE) VS. HIL(RED).....	255
FIGURE 166: ATTITUDE (COMPONENT 1 QUATERNION VECTOR PART), AIL(BLUE) VS. HIL(RED).....	255
FIGURE 167: ATTITUDE (COMPONENT 2 QUATERNION VECTOR PART), AIL(BLUE) VS. HIL(RED).....	255
FIGURE 168: ATTITUDE(COMPONENT 3 QUATERNION VECTOR PART), AIL(BLUE) VS. HIL(RED).....	256
FIGURE 169: ATTITUDE (COMPONENT 4 QUATERNION VECTOR PART), AIL(BLUE) VS. HIL(RED).....	256
FIGURE 170: CURRENT FLOWING WITHIN THE MTS	256
FIGURE 171: HIL SIMULATION FOR COMPLETE E-ST@R VERIFICATION – PROCESSES SETTINGS	258
FIGURE 172: HIL SIMULATION FOR COMPLETE E-ST@R VERIFICATION – MODELS FLOWS.....	259
FIGURE 173: HIL SIMULATION FOR COMPLETE E-ST@R VERIFICATION – C++ CODE SKELETON (I)	260
FIGURE 174: HIL SIMULATION FOR COMPLETE E-ST@R VERIFICATION – C++ CODE SKELETON (II)	262
FIGURE 175: COMPLETE HIL FOR E-ST@R FUNCTIONAL REQS VERIFICATION – ARRANGEMENT FOR THE TEST WITH QM	262
FIGURE 176: (A) BATTERY-BUS VOLTAGE AS A FUNCTION OF TIME, (B) 5V-BUS CURRENT AS A FUNCTION OF TIME, AND (C) 3.3V-BUS CURRENT AS A FUNCTION OF TIME	264
FIGURE 177: TELEMETRIES OF BATTERY PACK NO. 1 AS A FUNCTION OF TIME. (A) VOLTAGE, (B) CURRENT, AND (C) TEMPERATURE OF ONE CELL OF BATTERY PACK.....	264
FIGURE 178: SOLAR PANEL VOLTAGES. PANEL +X, PANEL +Y, PANEL –Y, PANEL +Z, AND PANEL –Z. PANELS ARE NAMED AFTER THE PERPENDICULAR VECTOR OF THE FACE THEY LIE ON, IN BODY AXES.....	265
FIGURE 179: ANGULAR VELOCITIES AS A FUNCTION OF TIME: (A) ANGULAR VELOCITY ALONG X AXIS, (B) ANGULAR VELOCITY ALONG Y AXIS, AND (C) ANGULAR VELOCITY ALONG Z AXIS	266

FIGURE 180: QUATERNION. (A) FIRST COMPONENT OF THE QUATERNION VECTOR PART, (B), SECOND COMPONENT OF THE QUATERNION VECTOR PART (C) THIRD COMPONENT OF THE QUATERNION VECTOR PART, (D,) QUATERNION SCALAR PART	266
FIGURE 181: REFERENCE FRAMES	274
FIGURE 182: EULER ANGLES REPRESENTATION	276
FIGURE 183: BLOCK DIAGRAM OF THE OPTIMAL LINEAR REGULATOR, USING RICCATI EQUATION.....	281
FIGURE 184: ADAPTIVE CONTROL STRUCTURE	282
FIGURE 185: BASIC MODEL OF A SINGLE NEURON.....	284
FIGURE 186: ACTIVATION FUNCTIONS	285
FIGURE 187: RD129 MICROPROCESSOR	286
FIGURE 188: RD126 DEVELOPMENT BOARD FOR RD129	287

List of Tables

TABLE 1: SYSTEM SIMULATION IN PRODUCT LIFE CYCLE [5]	23
TABLE 2: EPOS 2.0 MAIN FEATURES	33
TABLE 3: SCS INPUT/OUTPUT LIST	40
TABLE 4: FES INPUT/OUTPUT	42
TABLE 5: FVTB INPUT/OUTPUT	43
TABLE 6: SVF INPUT/OUTPUT	45
TABLE 7: AIV SIMULATOR INPUT/OUTPUT	47
TABLE 8: GSTS INPUT/OUTPUT	48
TABLE 9: TOMS INPUT/OUTPUT	49
TABLE 10: EXAMPLES OF EQUIPMENT/SUBSYSTEM MODELS	60
TABLE 11: EXAMPLES OF DYNAMICS AND ENVIRONMENTAL MODELS	60
TABLE 12: CE ACTIVITIES IN THE PHASES 0+A	73
TABLE 13: CE ACTIVITIES IN THE PHASE B	74
TABLE 14: CE ACTIVITIES IN THE PHASES C AND D	75
TABLE 15: CE ACTIVITIES IN THE PHASES E AND F	76
TABLE 16: GUIDANCE STRATEGIES FOR A SATELLITE	95
TABLE 17: GUIDANCE STRATEGIES FOR CHASERS SPACECRAFTS	95
TABLE 18: BASIC GUIDANCE STRATEGIES FOR LAUNCH VEHICLES	96
TABLE 19: MAIN ATTITUDE CONTROL METHODS [1]	96
TABLE 20: CONTROL STRATEGIES AND RELATED TECHNIQUES FOR SATELLITES	98
TABLE 21: CONTROL STRATEGIES AND RELATED TECHNIQUES FOR RVD/B	99
TABLE 22: CONTROL STRATEGIES AND RELATED TECHNIQUES FOR LAUNCHER AND RE-ENTRY VEHICLE	100
TABLE 23: EXTRINSIC AND INTRINSIC PERFORMANCES	100
TABLE 24: STARSIM'S LIST OF REQUIREMENTS	139
TABLE 25: SINGLE UNIX SPECIFICATION SIGNALS DEFINED IN <i>SIGNAL.H</i> [5]	146
TABLE 26: INTERFACES FUNCTIONS DATABASE	158
TABLE 27: MODELS DATABASE OF DEVICES AND EQUIPMENT	161
TABLE 28: MODELS DATABASES OF ENVIRONMENT AND S/C MOTION	161
TABLE 29: MODELS DATABASE OF GNC STRATEGIES AND SPECIAL FUNCTIONS	162
TABLE 30: TRANSFORMATION AND CONVERSION FUNCTIONS DATABASE	163
TABLE 31: MODELS DATABASE OF GSE	163
TABLE 32: MISSION PROFILE (REF. E-ST@R-II MISSION)	185
TABLE 33: OPERATIVE MODES	189
TABLE 34: MATRIX FUNCTIONS/COMPONENTS OF A-ADCS	196
TABLE 35: STARSIM IN THE E-ST@R LIFE CYCLE	198
TABLE 36: PID VS. LQR PERFORMANCES	207
TABLE 37: FUNCTIONAL AND OPERATIONAL REQUIREMENTS VERIFIED BY COMPLETE HIL SIMULATION	257
TABLE 38: MISSION AND ORBIT DATA	263
TABLE 39: PID	279

Acronyms

Acronyms	Meaning
AIV	Assembly Integration & Verification
ADC	Analog to Digital Converter
ADCS	Attitude Determination & Control System
API	Application Programming Interface
AR	Acceptance Review
ASSET	AeroSpace System Engineering Team
AIT/AIV	Assembly Integration & Test/Verification
CAD	Computer Aided Design
CAN	Controller Area Network
CDR	Critical Design Review
CDS	CubeSat Design Specification
COTS	Component Off The Shelf
DMU	Digital Mock-Up
DS	Deployment Switch
EKF	Extended Kalmen Filter
EM	Engineering Model
EMF	Earth Magnetic Field
FES	Functional Engineering Simulator
FM	Flight Model
FT	Functional Tree
FVT	Functional Validation Test-bench
FYS	Fly Your Satellite
GNC	Guidance Navigation and Control
GPIO	General Purpose Input Output
GPS	Global Positioning System
GIS	Ground (Control) Segment

GSE	Electrical Ground Support Equipment
GUI	Graphical User Interface
HIL	Hardware-in-the-Loop
ICD	Interface Control Document
IMU	Inertial Measurement Unit
I/O	Input Output
IPC	Inter-Process Communication
ISO	International Organization of Standardization
LAN	Local Area Network
LEO	Low Earth Orbit
LKF	Linear Kalman Filter
LV	Launch Vehicle
M&S	Modelling and Simulation
MBDV	Model Based Development and Verification
MCS	Mission Control System
MDR	Mission Design Review
MPS	Mission Performance Simulator
MT	Magnetic Torquer
MU	Mock Up
OBC	OnBoard Computer
OBSW	OnBoard SoftWare
OS	Operating System
PDR	Preliminary Design Review
PFM	Proto-Flight Model
PRR	Preliminary Requirements Review
PUS	Packet Utilisation Standard
PWM	Pulse-Width Modulation
QR	Qualification Review

RAM	Random Access Memory
RBF	Remove Before Flight
RF	Radio Frequency
RT	Real Time
RW	Reaction Wheel
S/C	Spacecraft
S&M	Structure & Mechanisms
SCS	System Concept Simulator
SE	System Engineering
SRR	System Requirements Review
STM	Structure and Mechanical (model)
SV	Singular Value
SVF	Software Validation Facility
SVT	System Validation Test
TC	TeleCommand
TLE	Two Lines Elements
TM	Telemetry
UKF	Unscented Kalman Filter
USART	Universal Serial Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
UTC	Coodinated Universal Time
V&V	Verification & Validation
WS	WorkStation

Definitions

Term	Definition
<i>Accuracy</i>	The closeness (of a model) to the real values.
<i>Algorithm</i>	A numerical representation of an items' behaviour – often used within a model of that item. Some algorithms operate on a network of entities (e.g. power and thermal models) and may be iterative.
<i>Calibration</i>	Validation of a model against another known reference (e.g. real data or another validated model).
<i>Component</i>	An implemented model with well defined interfaces that can be delivered in source or object code form. One or more instances can be instantiated within a single simulation. (See also portability)
<i>Domain</i>	Context of use in which a simulation and its models are intended. Common domains are e.g.: Operations, training, engineering (performance, testing), visualisation, ... (see also model)
<i>End-to-End Simulator</i>	This is used to simulate the end-product of a mission. It is also called a <i>Mission Performance Simulator</i> or <i>Functional Engineering Simulator</i> , depending on the mission.
<i>Failure¹ simulation</i>	The ability to model the failure of an item (or some other anomalous behaviour) in the simulator and to be able to fail this at run-time and also “unfail” or restore it. The model usually represents the effect of the failure, not necessarily the cause. The failures to be modelled are usually taken as a subset from the Failure Models Effects and Criticality Analysis.

¹ The term Failure is defined in ECSS-S-ST-00-01: the termination of the ability of an item to perform a required function

NOTE 1 After the failure, the item has a fault

NOTE 2 This concept as defined does not apply to items consisting of software only

Term	Definition
<i>Fidelity</i> ²	<p>How accurately a model represents the behaviour of the item or the environment it is modelling. Standard terms can help to define the fidelity requirements for a model:</p> <p><i>Accurate</i> Concepts that are modelled to a declared tolerance. Such tolerances should be stated explicitly. The normal values for telemetry parameters dependent upon this model should be within limits (if defined).</p> <p><i>Emulated</i> Simulating specifically processors allow the real software code/image to run inside the simulation.</p> <p><i>Exact</i> Used to describe concepts for which a zero tolerance is applicable. This is normally applicable to discrete systems.</p> <p><i>Functionally</i> Functionally modelled units/functions should work/behave as the real unit/function with respect to their external interfaces.</p> <p><i>Plausible or Realistic</i> Variables that should be modelled such that trends can be observed in their behaviour in relation to outside influence without being precisely modelled to a declared tolerance.</p> <p><i>Representative</i> Data described as representative does not need to be modelled; pre-set value should be provided within the measurement range of the parameter. This value will always be used by the simulator unless updated from the simulator console, when desired.</p> <p><i>Static</i> Fixed values only.</p>
<i>Hard Real-Time</i>	<p>One or more models must be executed within a certain time deadline. Specific guarantees are given about the duration of update periods. Typically a model will be assigned a “slot” in which it has to execute. Failure to do so may terminate the simulation. Mostly used in Hardware-in-the-loop simulations (See also Soft Real-Time).</p>
<i>Hardware-in-the-Loop</i>	<p>A simulation which is interfaced to external hardware – typically real or breadboard equipment. This is often used to support testing of the equipment.</p>
<i>Initialisation</i>	<p>The setting of the initial state of a model or simulation before a simulation run is started.</p>

² All fidelity requirements need to be verifiable for them to be of use.

Term	Definition
<i>Integration</i>	[1] In the domain of software engineering the joining of modules to form a complete system [2] In the domain of simulation the mathematical integration of state variables, usually over time
<i>Jitter</i>	For simulations Jitter is typically characterised by the short term-variations in the timing of a digital signal (e.g. a clock synchronisation pulse) (typically at 10Hz or greater). Below 10Hz, is termed Wander.
<i>Latency</i>	The time delay between the moment something is initiated, and the moment one of its effects begins e.g. between a command initiated to set an interrupt and the onboard computer responding to it.
<i>Model</i>	By simulation models it is meant here both data models, e.g. geometrical model of a system, and behavioural models, e.g. the algorithms representing the behaviour of a component or environment expressed in a high level programming language. A model normally (but not always) has inputs, outputs and internal state variables and constants. A generic model represents an entity (e.g. a power distribution network) that can be configured to represent any instantiation of that entity. Note: Although generic models are a powerful concept, they can become over complex and it becomes more effort to configure a generic model than to develop a specific model from scratch. Depending on the context, models can be classified according to their fidelity, their domain or their modelling technique.
<i>Models flow</i>	It is a graphical visualization of the models and the sequence in which it appears in the simulation loop.
<i>Modelling Technique</i>	Method used to analyse and describe the behaviour of a model. Common techniques are e.g.: Physical (electrical, mechanical...), behavioural, functional (with respect to external interfaces), geometric, ... (see also model)
<i>Portability</i>	The ability to use a model in different simulation environments, different hardware platforms and different operating systems, usually just by recompiling (compare interoperability).
<i>Precision</i>	The degree to which a measurement is made e.g. 3 significant figures.
<i>Real-Time</i>	A simulation in which the simulated time progresses at the same speed as the wall-clock time (but is not usually the same as wall clock time). See also Hard Real-Time and Soft Real Time.
<i>Restore</i>	The ability to load a saved simulation state and start a simulation run from the state where the simulation was saved. In most simulation environments, restore does not work if model variables have been added, removed, or have different types.

Term	Definition
<i>Save</i>	The ability to save the state of a simulation at a given instant in time. This is used to allow users to start a simulation from various predefined states e.g. Eclipse Entry (see also Restore).
<i>Savepoint</i> ³	The complete state of a simulation (the value of all its parameters and variables) which can be saved in such a way that a simulation can be restored to the same state and resumed at a later time. This is as well being referred to as: <i>Stateset, Breakpoint, Snapshot</i>
<i>Scenario</i>	A particular initial configuration of a simulator and sequence of events to represent a particular part of a mission e.g. launcher deployment, eclipse operations, cruise phase.
<i>Scheduler</i>	A component of the simulation environment responsible for scheduling the execution of the models within a simulator. The scheduler can schedule the models cyclically (usually at multiples of a base frequency) and/or according to asynchronous events (e.g. a telecommand arriving). (See as well continuous simulation and discrete simulation)
<i>Simulation</i>	A run of scenario in a simulator with a simulated start- and end-time. During the simulation events may be injected into the simulation by the user, a script, external hardware or another simulation.
<i>Simulation Environment</i>	The software infrastructure that is used to run models. It usually has a scheduler, supports the control of the models (via scripts and/or a GUI), visualisation of their public state variables and provides the simulation time. It may also provide other services such as save/restore, logging of model events and other events. Examples are EUROSIM, SIMSAT and SIMWARE.
<i>Simulator</i>	An ensemble of one or more models that are executed together to represent the behaviour of phenomena and/or an artificial system (e.g. spacecraft). It also includes the simulator kernel with the model scheduling.
<i>Soft Real-Time</i>	A real-time simulation in which the simulated time can slip without effecting the simulation results, with the expectation that recovers the slip later. (See as well Hard Real-Time)
<i>Stability</i>	The stability of the output from a model over time (or range a values).

³ Any alternative term can be proposed for “Savepoint” – legacy should be reflected in definition

Term	Definition
<i>State Variables</i>	Variables that represent a model's state at a moment in time. These may be public (visible to the simulation environment or other models) or private to the model itself. Note: These variables represent the (minimal) set of elements to be taken into account when proceeding Save and Restore actions.
<i>State Vector</i>	The ensemble of the state variables, relevant to be kept for a <i>Savepoint</i> .
<i>Test Facility</i>	The Test Facility is combined to the Product under Test to constitute the Test Platform. It generally consists of a Simulation Kernel, a Database, a Test Supervisor and Front Ends.
<i>Tuning</i>	The modification of model/simulator parameters to match as closely as possible to the expected data (part of validation).
<i>User Command</i>	A command that the user can initiate to change the state or behaviour of the simulator. Note: These are often used to set <i>failures</i> or to pre-configure the spacecraft to a certain state. They may also be used to change an operative mode or to start/stop monitoring and recording.
<i>Validity</i>	The range over which a model/simulation is valid (e.g. due to certain assumptions in the algorithms or integration time-step used).

Introduction

The research presented in this thesis can be framed in the field of applied research, intended as the application of scientific knowledge and engineering methods to solve practical problems. We propose a solution to support the engineering team during the design and verification process of a space system. The work done aims at improving the design and verification activities through an approach that increases the effectiveness and efficiency of these processes. This approach foresees the definition, development and validation of a methodology and related tool to be used by designers, developers, analysts and operators during different phases of the product life cycle.

In the last few years, systems have become more and more complex due to several factors. From the technical point of view, the number of functions implemented in a single system is continuously increasing thanks to the dramatic progress of technology in the last decades. At the same time, the relationships between functions and hardware of the system, as well as the interactions between different elements and disciplines which concur to its definition, are growing up in number and gaining in complexity.

System Engineering (SE) is the discipline which ensures that any kind of requirement within a program frame is satisfied avoiding that some aspects prevail on others and promoting a reduction of costs, time and resources without losing effectiveness and reliability. SE is based on an iterative and recursive process of management, design, development, construction, verification, starting from the system level and going down through subsystem level and components level.

Within this context, a new approach is taking shape: the model-centric system engineering, or model based system engineering (MBSE). It is substituting the document-centric approach because it allows a more efficient management of the information flow and data repository among actors with different technical skills and organization and, in many case, geographically distributed. MBSE is increasingly making use of the Model and Simulation (M&S) approach, which is replacing the superseded “design-produce-test” approach. This is especially true in the space industry, where the traditional approach of testing after production is ineffective for several reasons. First, it is very expensive in terms of resources (budget and time) needed to get the real hardware. Second, a minimum defect in the hardware (due to design or manufacturing) might jeopardize the whole test session thus delaying the project schedule. Moreover, it is sometimes impossible to test some features due to the special environment (Space) in which the system will operate. Simulation seems to be the perfect means to “test” space systems and it can be used throughout the product life-cycle. In particular, M&S based approach allows verifying solutions before hardware manufacturing.

On the other hand, on board software and hardware functionalities cannot be verified and validated utilizing only “pure” simulation because it may not be exhaustive. In fact, notwithstanding the modeling effort, pure simulation alone gives only part of the answer to the problem and verifications on actual software and hardware are necessary. The real system may exhibit behaviors which cannot be modelled perfectly and that affects the outputs to a great extent, for example delays and uncertainties are not well predictable by models. For this reason, verification sessions with real on board software, controller/processor and hardware in the loop are necessary. Often the literature does not address the interdependency among algorithms, software, controller/processor, and hardware, but, according to Eickoff, it results “essential for understanding which participant in a project at which time is dependent on which input results. Or, formulated vice versa, who in the project will be pushed onto the critical path in development by delayed input.”

Taking into account that the verification of any functionality and performance through the tests is repeated along the life cycle, an “infrastructure” is needed, independently from the complexity level of

the test and the development stage of the project. This infrastructure is a design and test environment that must be suitable for evaluating 1) the algorithms, 2) their implementation in the software, 3) the functionality of the controller, and 4) the integration of the whole final system. We call this infrastructure “simulator”.

The activities described in this thesis deal with the development of a simulation platform able to support the engineering team throughout the life-cycle of a space product. Two main features of this simulator are worth mentioning, namely portability and flexibility. As far as the former is concerned, it is important that the simulation results are portable from one stage of development to the next. It means that great attention shall be paid to avoid that information are lost and that the format of data are compatible through all phases. Incompatibility of the results is quite typical in many tools devoted to only one phase of the life cycle or too general purposes. Today, the tendency is putting together different software and facilities to cover the whole life cycle process requiring great efforts to guarantee the compatibility and fidelity among all the elements: the exchange of information/data but also requirements, documentation shall be extremely precise in order to avoid misunderstanding. But, in any case, it is impossible to be extremely confident in this process in particular when the project passes from a tool/instrument/software to another through the various phases. Using a single platform helps to ensure portability, but requires that the simulator is flexible enough to adapt to each stage of the life-cycle.

The solution proposed in this thesis is a single simulator based on free software and firmware and low cost GSE which can be configured as needed according to the life-cycle phase of interest.

The simulation facility has been developed specifically for the design and verification of space systems, but it could be extended in principle to other applications. In particular, the activity has been focused on the design and verification of Guidance Navigation and Control (GNC) Systems. The theme was proposed by Aviospace srl, one EADS-Astrium subsidiary in Italy. They were interested in the acquisition of knowledge, methodologies and tools for the design and verification of GNC systems to be used onboard future space vehicles. The AeroSpace System Engineering Team (ASSET) in the Department of Mechanical and Aerospace Engineering (DIMEAS) at Politecnico di Torino has been working for years on the definition and implementation of effective and efficient methodologies for supporting the design and verification process of complex systems for aerospace applications.

The interests of Aviospace, the expertise of ASSET and the background of the candidate, having a Master Degree in Computer Science and Automatic Control Engineering and work experiences in the aerospace field, provided the rationale for a PhD research aimed at investigating new solutions to improve the design, verification and validation processes of a specific system, namely the GNC.

The choice of the GNC system was born, as said before, from common interests and background of the partners involved in the research, but other aspects have influenced the choice. First consideration is that the GNC system is a vital system of whatever spacecraft, and many reliability- and safety-related requirements apply. Second, it is often characterised by a high intrinsic complexity as well as a high degree of interaction with other spacecraft subsystems and the mission as a whole. Both specialist’s knowledge and system engineer’s skills are required to design and develop a successful GNC system. In fact, GNC subsystem strongly affects and is affected by the design and the verification strategies of the entire system. Third, it is usually a very expensive subsystem, for both the technology involved and the mass associated to its implementation. Last but not least, GNC system’s behavior is difficult to be verified on ground because limitations exist in reproducing the space environment. For all the abovementioned reasons, we think that the study of new and cheap solutions able to improve the confidence level of the design and verification result is a challenging research and will be still a point of interest and discussion in the space field in the future.

The efforts have been addressed to GNC subsystems for space vehicles in well-defined and specified missions. In particular, three types of reference missions are taken into account. The first case relates

to educational and scientific missions carried out by small satellites, as an example of a single orbiting object mission. The second mission is a Rendez-Vous and Docking mission with non-collaborative target to represent the case of two objects that should mate in orbit. Last example is a launch missions performed by a space transportation vehicle to represent the case of one object that should reach the orbit. These cases have been chosen because they present different features and criticalities from the GNC point of view in terms strategies, functional architectures, hardware and software. They allow to investigate more and various aspects of GNC systems design and verification. However, analyzing the state of the art referred to GNC subsystem, common functions can be found but different control modes and strategies are implemented; sensors and actuators make use of several technologies according to the required performance but basic concepts of safety, reliability, and level of autonomy are common baselines in any project. Summarizing, the chosen space applications are considered an exhaustive representative set for the major characteristics of the GNC system and they are good case studies and test benches for the proposed methodology and the simulator validation.

This thesis is focussed on the definition of the simulator and underlying methodology. Although all three reference missions have been studied, only the first case has been completed to full validation of the project, i.e. till the Hardware-in-the-Loop simulation. The second and third cases would have required resources well beyond the scope and possibility of a doctoral program. Some of the research and industrial programs within which the methodology is applied are still ongoing and will continue for a long time after the conclusion of the present research. However a first significant version of the simulator has been released within the PhD period. The work done lays the foundation for future activities in terms of development and upgrade of the tools for the project of other space systems' GNC and optimization of the processes according to the proposed methodology. In conclusion, we can say that a HIL simulator has been designed and implemented for a case of interest to demonstrate the feasibility of the approach.

Structure of the thesis

The thesis is divided in five chapters.

Chapter 1 proposes a survey on the M&S based approach in MBSE discipline. Analogies and differences with respect to a traditional document-based approach are shown. The concepts of design, verification and validation are investigated and inserted in the M&S based approach. Simulation configurations and strategies are explained into the details and the concept of model is investigated, underlining the main features and parameters for a categorization in databases. Finally, an overview on the state of the art of methods, tools and facilities for the verification of space systems is presented.

Chapter 2 is devoted to GNC systems. The state of the art related to the GNC with respect to missions and vehicles allows to trace GNC features: the nominal and off nominal control modes, the architectures, the navigation algorithms, the guidance strategies, the control methods and techniques and performances evaluation are treated. Criticalities and possible solutions for the GNC system design and verification are assessed in order to derive guidelines useful during the preliminary studies of the methodology.

Chapter 3 is the heart of the work: the design and development of the simulator (namely StarSim) is presented. The StarSim requirements are derived from an accurate analysis of the needs that a simulator shall satisfy and the tasks that the simulator shall accomplish. The functional architectures and the different simulation configurations (AIL, SIL, CIL, HIL and Hybrid) are shown. Each element of the simulator is deeply described and many examples support the demonstration of the capabilities and the features. To conclude a summary of the user manual for the first release is provided.

Chapter 4 presents the case study on which the methodology and the simulator are applied: the e-st@r CubeSat program. It is an educational project based on the complete design (from the feasibility study to the operations in orbit) of a satellite belonging to the CubeSat standard. The Active Attitude

Determination and Control System (A-ADCS) is the payload of the first two 1U satellites. The simulator has been used for the design and verification of the A-ADCS in a number of configurations. StarSim covers the feasibility study, through AIL simulation, during which the preliminary sizing of the system is accomplished and mission parameters and constraints are evaluated. In the design and development phase alternative architectures have been evaluated, determination and control algorithms have been defined (through AIL simulation), the software and its performance have been tested (via SIL and CIL simulation), and the design of the A-ADCS board has been carried out and tested, including calibration of sensors and actuators and verification of extrinsic performances. Finally, in the integration and qualification phase, through HIL and Hybrid simulations, the entire system, i.e. the CubeSat, has been tested against high level requirements.

In the concluding section, Chapter 5, the work done is assessed highlighting the results obtained and the road for future improvement and upgrade of the StarSim simulator is traced.

Chapter 1. Model and Simulation Based approach in System Engineering

1.1 System Engineering & Programs Management methodologies

The execution of complex programs, as space programs, requires the contemporary activity of several people and organizations with the common objective of delivery a final product that satisfies the customer performance needs within the agreed schedule and budgets. For this purpose, technical activities as well as human and financial resources shall be organized and coordinated in a structured way in order to reach the objectives.

In the context of the overall program management, System Engineering (SE) plays an important role. SE is a managerial and technical process that allows to develop operable systems capable of meeting requirements within imposed constraints. The main goal is to evaluate and balance the contributions of structural engineers, electrical engineers, mechanism designers, power engineers, human factors engineers, and many more disciplines one against another, to produce a coherent whole that is not dominated by the perspective of a single discipline. The system engineer focuses efforts on assessments to optimize the overall design, and not make more attention to one system/subsystem than any others. Moreover, the systems engineer skills deal with the balancing of technical, managerial/organizational, economical, logistic, and relationship issues searching the best compromise and tradeoffs among specialists and generalists.

System engineering strategy can be seen as a spiral approach (Figure 1) that starts with the mission needs and system requirements definition. After that, a risk assessment is performed and several system models are implemented in order to verify and qualify it.

This is an iterative approach since the components integration and verification tasks could lead to a system design review, which means to define or reconsider system architecture or even mission objectives. The spiral process allows reaching a compromise between the mission needs, conveyed by the client or contracting owner, and the system performances and functions. The iterative approach ensures the objectives to converge along with a progressive verification of the technical resources and state of the art.

Given a well-defined mission, SE approach consists of identifying related objectives and needs, which should be broken into elementary ones. System engineer define the right functional architecture which allows integrating the elementary solutions in order to reach the mission needs. Each elementary need is satisfied by an elementary sub-system. Once the system engineer has defined the system requirements and its functional architecture, system integration task is accomplished. Integration engineering philosophy allows to assemble, not only physically but also functionally, several sub-systems and to verify their functions as well as their interfaces (mechanical, electrical, data exchange) and with the environment where the system will be exploited. The key point is to manage heterogeneities. All sub-systems shall work properly, despite their differences and heterogeneities, and shall interface to the others in order to accomplish mission objectives as a whole homogeneous system.

The focal actor for the system engineer is the system. System can be explained as a nucleus of elements structured in such a manner as to accomplish a function to satisfy an identified need. The elements, or parts, can include people, hardware, equipment, software, facilities, policies, and documents; that is, all things required to produce system-level results. All the involved elements are required for the preparation, the operation and/or the support of the system. The elements shall

organized in a well defined and structured way such as the system results a construct or collection of different elements that together produce results not obtainable by the elements alone. The value added by the system as a whole, beyond that contributed independently by the parts, is primarily created by the relationship among the parts, in other words, how the parts are interconnected. This stresses how much important are the interfaces between sub-systems, to which the greatest effort of system engineers is committed.

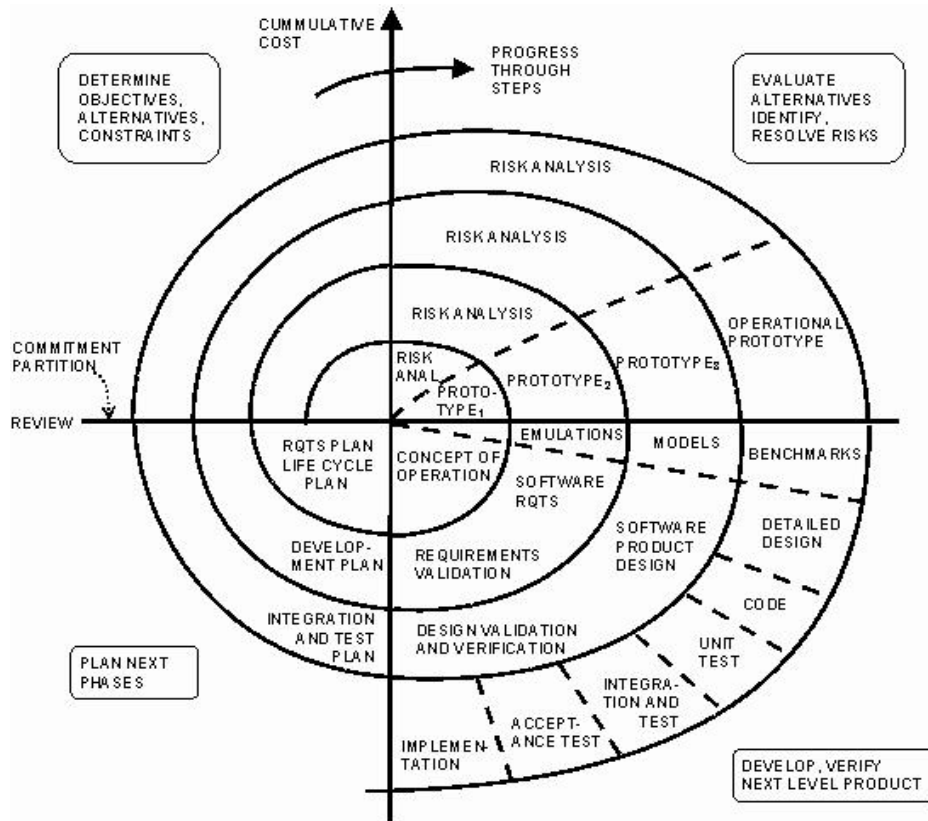


Figure 1: SE spiral approach (BOHEM 1988)

Definitely, system is the sum of the elements and the relationship among them: designing and implementing a system means to create a surplus value, which means to perform a function that single components are not able to accomplish.

Each system can be decomposed into smaller components. For example, for the space segment, a space vehicle can be broken up into systems, sub-systems, equipments and components, according to a descending hierarchy. The system definition given before has a broader scope since the strategies of system engineering, integration and verification shall be exploitable not only at system level, but also at the lowest level at which a system could be decomposed. At the same time, a system can be part of a bigger system (system of systems) constituted by a series of units having the same objectives and function to accomplish (e.g. a satellites constellation) and related one among the other not only from a technical point of view but also for programmatic aspects.

Pay attention that system definition changes according to the context. An example is an Inertial Measurement Unit (IMU). For the sensor producer, IMU is the system constituted by the accelerometers, gyroscopes, electrical circuits, case, and I/O connections. For a spacecraft designer, IMU is a component of the GNC system of the spacecraft. For the customer of the spacecraft, it is a component of a wider global system constituted also by facilities, personnel training, user's services, maintenance services. In this thesis the focus is the GNC System for different kinds of space vehicle: it is a particularly complex subsystem of a system (the vehicle) constituted by other subsystems (OBC,

EPS, Propulsion, TCS, COM SYS, Structure and Mechanics, TCS/TPS and so on) with which the GNC has mechanical, electrical and logical interface and that influence the subsystem design as well as the GNC condition the other subsystems and the entire vehicle project. GNC is also constituted by sensors, actuators and elaboration units that interact in order to perform in a satisfactory way the navigation, guidance and control tasks. See the Chapter 2 for a complete analysis on GNC.

1.1.1 The space product life cycle phases

SE process is strictly related to the product life cycle that, according to the [1], is typically divided into seven phases, as shown in Figure 2:

1. Phase 0: Mission analysis and needs identification
2. Phase A: Feasibility
3. Phase B: Preliminary definition
4. Phase C: Detailed definition
5. Phase D: Production and Qualification
6. Phase E: Utilization
7. Phase F: Disposal

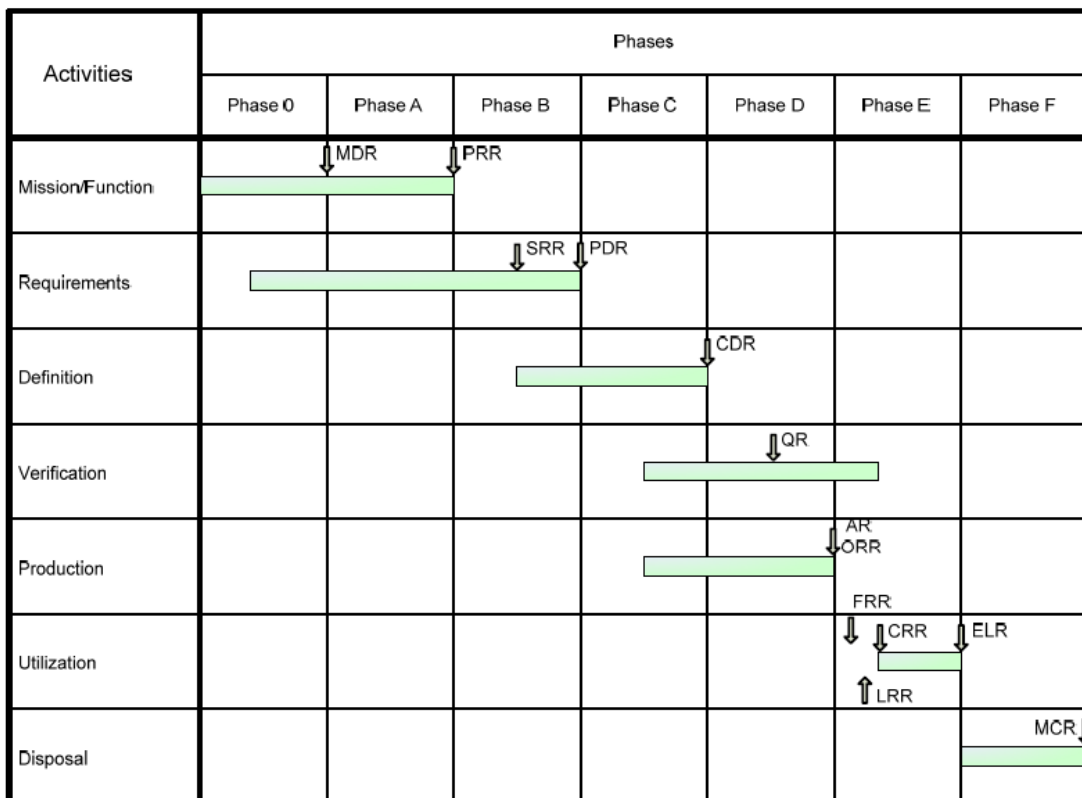


Figure 2: Product Life Cycle

Before going into details, project phases are closely linked to activities on system and product level and, depending on the specific circumstances of a project and the acceptance of involved risk, activities can overlap project phases.

Phases 0, A, and B are focused mainly on:

- the elaboration of system functional and technical requirements and identification of system concepts to comply with the mission statement, taking into account the technical and programmatic constraints identified by the project initiator and top level customer.
- the identification of all activities and resources to be used to develop the space and ground segments of the project,

- the initial assessments of technical and programmatic risk,
- the initiation of pre-development activities.

Phases C and D comprise all activities needed to develop and qualify the space and ground segments and their products.

Phase E comprises all activities to be performed in order to launch, commission, utilize, and maintain the orbital elements of the space segment and utilize and maintain the associated ground segment.

Phase F comprises all activities to be performed in order to safely dispose all products launched into space as well as ground segment.

- Phase 0 - Mission analysis/needs identification

SE team works together the project initiator, the top level customer and representatives of the end users in order to identify and characterize the mission needs, expected performance, dependability and safety goals and mission operating constraints with respect to the physical and operational environment. All these aspects shall emerge in the mission statement from derive the preliminary technical requirements specification and possible mission concepts. Preliminary risk assessment is performed. At the end of the phase a mission definition review (MDR) is held and the outcome of this review is used to judge the readiness of the project to move into phase A.

- Phase A - Feasibility

The feasibility phase establishes the preliminary management plan, system engineering plan and product assurance plan for the project. SE team elaborates and compares against the need possible functional system architecture and operations concepts.

Moreover, it assesses the technical and programmatic feasibility of the possible concepts by identifying constraints relating to implementation, costs, schedules, organization, operations, maintenance, production and disposal. For each possible solution it quantify and characterize critical elements and technology in order to support the decision making about the system and operations concept(s) and technical solutions, including model philosophy and verification approach, to be further elaborated during Phase B. At the end of the phase A, the preliminary requirements review (PRR) is held at the end of phase in order to judge the readiness of the project to move into Phase B. The primary objectives of this review are:

- Release of preliminary management, engineering and product assurance plans.
- Release of the technical requirements specification.
- Confirmation of the technical and programmatic feasibility of the system concept(s).
- Selection of system and operations concept(s) and technical solutions, including model philosophy and verification approach, to be carried forward into Phase B.

- Phase B - Preliminary Definition

Phase B foresees the system design refinement and verification. SE team elaborates cost analysis, organize schedule, tasks and milestone in the Work Breakdown Structure (WBS). Through technical trade-off” studies, the preferred system concept, together with the preferred technical solution(s) for this concept is selected. Another big task is the determination of the verification program including model philosophy. Reliability, safety and risk assessments are conducted and, finally, space debris mitigation and disposal plan are prepared.

There are 2 project reviews associated with Phase B.

- The system requirements review (SRR) held during the course of Phase B.
- The preliminary design review (PDR) held at the end of Phase B. The outcome of this review is used to judge the readiness of the project to move into Phase C.

The primary objectives of SSR review are:

- Release of updated technical requirements specifications.
- Assessment of the preliminary design definition.
- Assessment of the preliminary verification program.

The primary objectives of PDR review are:

- Verification of the preliminary design of the selected concept and technical solutions against project and system requirements.
- Release of final management, engineering and product assurance plans.
- Release of product tree, work breakdown structure and specification tree.
- Release of the verification plan (including model philosophy).

- Phase C - Detailed Definition

The scope and type of tasks undertaken during this phase are driven by the model philosophy selected for the project, as well as the verification approach adopted.

The main task is the detailed design definition of the system at all levels in the customer-supplier chain.

Production, development testing and pre-qualification of selected critical elements and engineering models, as required by the selected model philosophy and verification approach, start. AIT planning for the system and its constituent parts is produced.

The critical design review (CDR) is held at the end of phase C. The outcome of this review is used to judge the readiness of the project to move into phase D. The main review objectives

- Assess the qualification and validation status of the critical processes and their readiness for deployment for phase D.
- Confirm compatibility with external interfaces.
- Release the final design.
- Release assembly, integration and test planning.
- Release flight hardware/software manufacturing, assembly and testing.
- Release of user manual.

- Phase D - Qualification and Production

Phase D foresees mainly the completion of

- qualification testing and associated verification activities.
- manufacturing, assembly and testing of flight hardware/software and associated ground support hardware/software.
- the interoperability testing between the space and ground segment.

There are 3 project reviews associated with phase D

- The qualification review (QR) held during the course of the phase. The primary objectives of this review are:
 - To confirm that the verification process has demonstrated that the design, including margins, meets the applicable requirements.
 - To verify that the verification record is complete at this and all lower levels in the customer-supplier chain.
 - To verify the acceptability of all waivers and deviations.

Where development encompasses the production of one or several recurring products, the QR is completed by functional configuration verification during which:

- The first article configuration is analyzed from the viewpoint of reproducibility.
 - The production master files for the series productions are released.
 - The series production go-ahead file is accepted by the customer.
- The acceptance review (AR) held at the end of the phase. The outcome of this review is used to judge the readiness of the product for delivery.

The primary objectives of this review are:

- To confirm that the verification process has demonstrated that the product is free of workmanship errors and is ready for subsequent operational use.
 - To verify that the acceptance verification record is complete at this and all lower levels in the customer-supplier chain.
 - To verify that all deliverable products are available per the approved deliverable items list.
 - To verify the “as-built” product and its constituent components against the required “as designed” product and its constituent components.
 - To verify the acceptability of all waivers and deviations.
 - To verify that the Acceptance Data Package is complete.
 - To authorize delivery of the product.
 - To release the certificate of acceptance.
- The operational readiness review (ORR), held at the end of the phase. The primary objectives of this review are:
 - To verify readiness of the operational procedures and their compatibility with the flight system.
 - To verify readiness of the operations teams.
 - To accept and release the ground segment for operations.
- Phase E –Utilization

The major tasks for this phase vary widely as a function of the type of project concerned. They foresee:

- to prepare the space and ground segment for the launch
- to conduct all launch and early orbital operations, including the commissioning activities.
- to perform the on-orbit and ground operations in order to achieve the mission objectives.
- to provide the final disposal plan.

There are 4 project reviews associated with phase E.

- The flight readiness review (FRR) is held prior to launch. The flight readiness review is conducted prior to launch. The objective of this review is to verify that the flight and ground segments including all supporting systems such as tracking systems, communication systems and safety systems are ready for launch.
- The launch readiness review (LRR), held immediately prior to launch. The launch readiness review is conducted just prior to launch. The objective of this review is to declare readiness for launch of the launch vehicle, the space and ground segments including all supporting systems such as tracking systems, communication systems and safety systems and to provide the authorization to proceed for launch.

- The commissioning result review (CRR), held after completion of the on orbit commissioning activities. The commissioning result review is held at the end of the commissioning as part of the in-orbit stage verification. It allows declaring readiness for routine operations/utilization. This Review is conducted following completion of a series of on-orbit tests designed to verify that all elements of the system are performing within the specified performance parameters. Successful completion of this review is typically used to mark the formal handover of the system to the project initiator or to the system operator.
The end-of-life review (ELR) held at the completion of the mission. The main objectives are:
 - To verify that the mission has completed its useful operation or service.
 - To ensure that all on-orbit elements are configured to allow safe disposal.
- Phase F – Disposal in which the disposal plan is implemented.
The mission close-out review (MCR) is held at the end of this phase. The review objectives is to ensure that all mission disposal activities are adequately completed.

1.1.2 V-model

Each of the above project phases includes end milestones in the form of project review(s), the outcome of which determines readiness of the project to move forward to the next phase.

With the exception of the MDR which normally involves only the project initiator, and the top level customer, all other project reviews up to and including the AR are typically carried out by all project actors down to the lowest level supplier in the customer-supplier chain involved in the project phases containing these reviews.

From the PRR to the PDR, the sequence of the reviews is “top down”, starting with the top level customer and his top level supplier, and continuing down the customer-supplier chain to the lowest level supplier. From the CDR to the AR, the sequence of reviews is reversed to “bottom up”, starting with the lowest level supplier and its customer and continuing up through the customer supplier chain to the 1st level supplier and the top level customer. This is the so called “V model”.

The V-model is a graphical representation of the systems development lifecycle, which can be applied to hardware as well as to software systems. Instead of moving down in a linear way, the process steps are bent upwards after the sub-systems integration phase, resulting in the typical “V” shape.

This model stands on the relationship between each phase of the development life cycle and its associated phase of testing. The horizontal and vertical axes represents time or project completeness (left-to-right) and level of abstraction, respectively.

More in detail, the left side of the “V” reports the decomposition of requirements and creation of system specifications, from system level to components level. Whereas, the right side of the “V” represents integration of parts and their verification and validation, from components to system level.

The “V” model allows minimizing the project risks, improving the project transparency and project control by specifying standardized approaches and describing the corresponding results and responsible roles. It permits an early recognition of planning deviations and risks and improves process management, thus reducing the project risk. Moreover, the “V” model ensures the quality and the completeness of the results: the effort for the development, production, operation and maintenance of a system can be calculated, estimated and controlled in a transparent manner by applying a standardized process model.

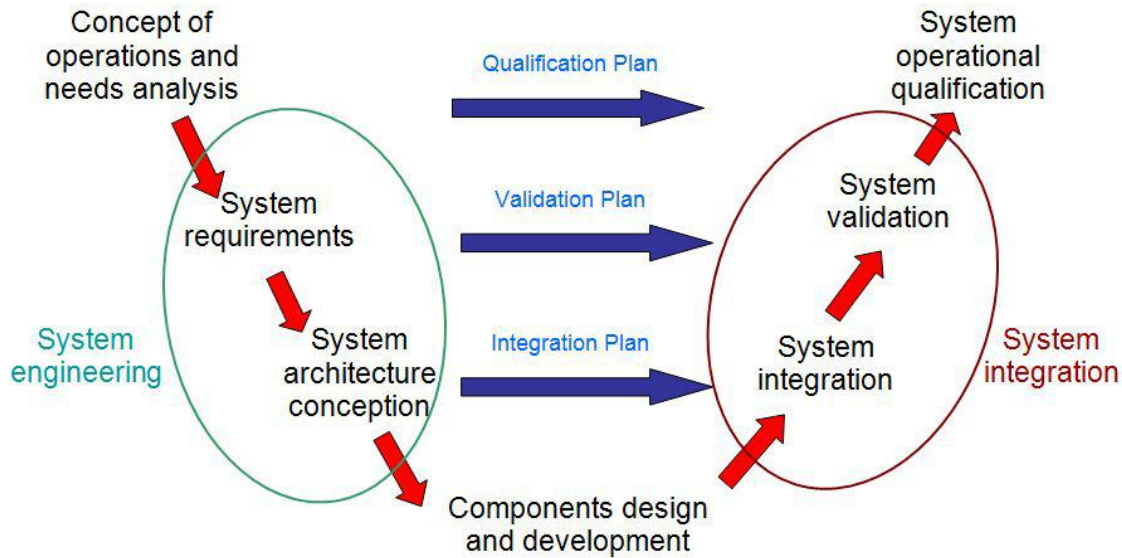


Figure 3: Traditional V-model in SE

1.2 Focus on verification and validation

The verification of an item, element, function, subsystem, system, process or product shows proof of compliance with requirements, that the product can meet each “shall” statement as proven through performance of a test, analysis, inspection, or demonstration. [2] Verification testing relates back to the approved requirements set and can be performed at different stages in the product life cycle. The approved specifications, drawings, parts lists, and other configurations documentation establish the configuration baseline of that product, which may have to be modified at a later time. Without a verified baseline and appropriate configuration controls, later modifications could be costly or cause major performance problems. Moreover, verification concerning the effects of orbital environment, such as thermal vacuum, radiation and launch loads, will be addressed only if their effects causes a change in system key parameters.

The validation of an item, element, function, subsystem, system, process or product shows that the product accomplishes the intended purpose in the intended environment, as well as the description by the mathematical modeling represents, to a “sufficient” accuracy, the behavior maintained by item, element, function, subsystem, system, process behave as expected under the real world conditions. Validation shall confirm that the object under verification meets the expectations of the customer and other stakeholders as shown through performance of a test, analysis, inspection, or demonstration [2], [3]. Validation testing is conducted under realistic conditions (or simulated conditions) on end products for the purpose of determining the effectiveness and suitability of the product for use in mission operations by typical users.

[4] says: “if a developer designs a system that complies with the specifications, but presents logical bugs, the system would fail the verification but successfully pass the validation. Conversely, if the system design is bug free but does not behave as expected, the model would fail the validation even though it passes the verification. In more common terms, the main purpose of V&V is to answer two key questions: (1) “Are we building the system right?” (verification) and (2) “Are we building the right system?” (validation)”.

In general, the verification and validation processes contribute to gaining a confidence level as high as possible of a system before the flight: in orbit performances and capabilities, according to the mission objectives and the design, shall be reached by way of analysis, simulation and testing. Going into the details, V&V process shall

- demonstrate the product design and performance, as meeting the specified requirements at the specified levels;
- ensure that the product is in agreement with the qualified design and it is free from workmanship defects and acceptable for use;
- confirm product integrity and performance at particular steps of the project life cycle.

Unfortunately, a lot of the on-orbit conditions cannot be reproduced on ground. Sometimes, facilities can reproduce partially some aspects of the general conditions, but they cannot be exhaustive for global considerations and they are quite expensive in terms of cost and resources to employ: in fact, verifying and validating is a controversial issue, mainly because a formal verification and validation campaign is highly expensive. Taking into account these issues, the goal for all verification and validation efforts can never be the achievement of absolute proof, but rather the acquisition of the highest possible level of confidence that a system or an operation will perform as required in the real mission under the real conditions.

Clearly, the maximum confidence about a system in the real mission derives from the testing prior to flight if it is subject to the operative conditions that it will experience during the mission. In particular for the testing the worst-case conditions of this environment need to be taken into account. Wherever possible, such functions and items should therefore be tested in a realistic physical environment thanks to specific, sophisticated and expensive facilities, including sufficient margins to cover such worst-case conditions. The method generally applied is to test maximum and minimum values along with a certain representative number of combinations of such values. This is, however, not the same as testing the entire field of possible variations and combinations, as it leaves the possibility of unidentified harmful combinations. Moreover, even if it were technically feasible to test the system, item or operation involving all environmental conditions relating to the mission, it would for reasons of time and cost be impossible to test all the potential variations and combinations of parameters and all possible contingency situations. An example of using specific facilities can be done. For a spacecraft, the essential zero gravity conditions cannot be provided on Earth. Therefore it theoretically is necessary to work with test stands where attitude and position of the spacecraft can be modeled geometrically. On three-dimensional turn table test stands with mounted spacecraft sensors, e.g. Sun and Earth visibilities for each sensor formerly were simulated by optical and infrared lamps. Similar setups existed for optical injection of star positions into star trackers. On these turn tables, at the same time, the angular rate sensors of the spacecraft could be stimulated. The cost to design, build and arrange sophisticated and accurate physical tools, validate the capabilities and maintain the entire facility would be high because the complexity is very high. However, the complexity even increases if a comparable approach is to be followed for additional attitude sensors, rotational rate sensors and finally also the for satellite's actuators - eventually even the pyrotechnic ones. Including all this hardware equipment in a closed-loop verification test stand is far out of financial mission budgets today.

For functions and performances as well as operations which cannot be verified by direct physical tests on ground, two other ways of verification are available in principle: mathematical modeling and simulation or testing in orbit under the conditions of the real flight.

Verification by testing in orbit is quite limited for reasons of launch cost and opportunity and because generally the complete mission is practically duplicated. In the best case, testing in orbit can be performed under similar conditions which must be proven to be sufficiently representative of the real mission.

For the majority of all features (e.g. orbit dynamics, microgravity effects, magnetic field and space environment), the verification campaign can be led through tools and facilities containing mathematical and stochastic modeling. For this reason, detailed mathematical models have to be established of the spacecraft, its dynamics and kinematics, of the actuators, of the sensors, of the

onboard data management system, of the communications links and equipment, and so on. This modeling must include all the (significant) effects that the orbital environment has on these features. To make them suitable for use in verification tools, these mathematical models need to be validated with respect to the according properties and effects of the real world, which are set by the spacecraft design and by the orbital environment. More detailed explanation on model philosophy is in paragraph 1.5.

In any case, for verify and validate a system, it will be to provide evidence that the representation of the reality by the model is correct and sufficiently complete for the purpose of verification. These requirements will be the most difficult one to fulfil, since it requires complete knowledge of the ‘real world’ with all its facets, knowledge that even with long experience will never be 100% complete. In other words, even after the most rigorous verification and validation process, uncertainty and risk will always remain.

For this reason, at least for unmanned spacecraft, a more pragmatic approach shall be followed based on simulation technologies as they are treated along this thesis.

1.2.1 Verification and validation in the space product life cycle

Verification and validation are processes useful in any phase of the product life cycle and they are not constrained to the end of a project (e.g., the *qualification phase*, during which it will be proven that everything is functioning and performing according to the requirements under all conditions of the real mission).

This initial consideration leads to review the V model because the verification and validation activities start already when the system and surely its subsystem and component are not defined. Figure 4 shows the new V model representation.

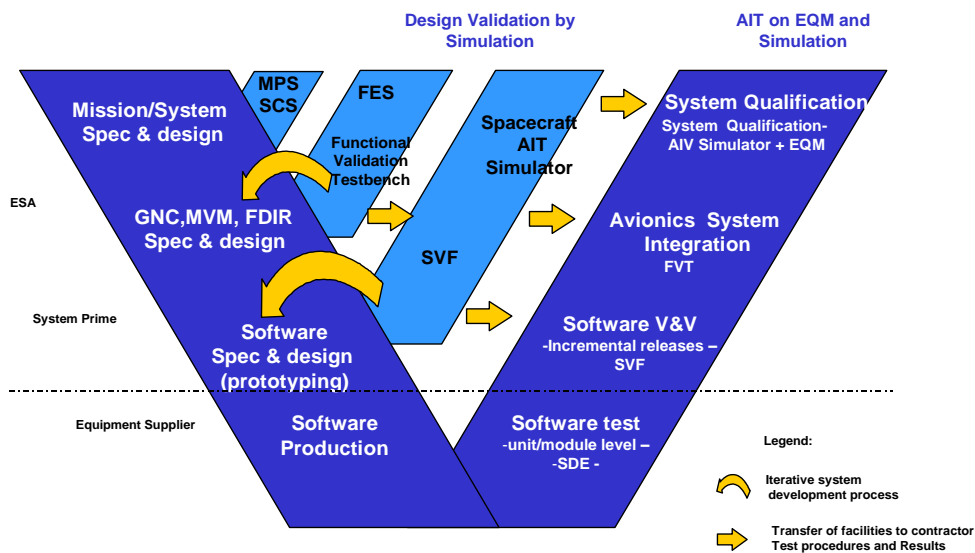


Figure 4: V model using Model and Simulation based approach for the verification [9]

Clearly the features, the methods, and the techniques of verification and validation in each phase have to be chosen because the objectives vary for each phase along the space product life-cycle. In this context, some aspects shall be taken into consideration:

- the design peculiarities and constraints,
- the qualification status of candidate solutions (product category),
- the availability and maturity of verification tools and facilities,

- the verification methods,
- the programmatic constraints,
- the cost and schedule.

The requirement criticality, in terms of technical and programmatic impacts on the verification implementation, should be assessed by the involvement of the verification team in the requirement definition process during phases A and B, since it drives the verification strategy. The verification approach should allow:

- To ensure the definition of correct verification criteria for each requirement by participating in the preparation of product specifications.
- To assess the impact that verification has on the design (e.g. modularity, testability, and accessibility).
- To ensure a coherent approach to verification implementation throughout the various levels avoiding duplication of activities.
- To ensure early verification of critical items to reduce the risks of late failure identification.
- To ensure the coverage of the interface verification.
- To optimize the design and use of ground support equipment, simulators, test tools and test software (e.g. re-use between levels, stages and models).
- To optimize the use of test facilities.
- To plan for feedback to the verification activity from the commissioning results in case of multi-mission projects or recurring products.
- To consider innovative solutions that can reduce overall verification costs.
- To provide visibility and objective evidence of verifications performed.

Test severity (stress level, environment, duration, etc.) shall be adjusted to requirements, design and verification strategy. Furthermore, they shall take into account test conditions (i.e. tolerances):

- Qualification test level (mechanical, thermal, lifecycle) are stricter compared with ground, launch and cruise expected constraints, in order to show design, fabrication or integration faults. The aim is to verify margins defined by analysis.
- Flight acceptance review reproduces ground, launch, cruise and space environment constraints, which means that the same test as the flight acceptance review are performed, although at a lower stress level and for a shorter time. The goal is to identify eventual material and process defaults.

Verification strategy shall be chosen taking into account the critical aspects of requirements to be verified and cost/delay compromise. For each verification task, pass/fail criteria shall be defined and expressed such that they are measurable and not ambiguous.

Requirements traceability is the prerequisite to every verification tasks. Hereafter, the implementation of verification tasks run through some formal documents: verification matrix associated to requirements, verification plan at system level or lower, system support specifications, Assembly Integration and Test (AIT) procedures and test reports. More in detail, verification matrix shows a set of information about verification strategies or their combination, test levels and their multiple steps, the number of physical models verified. This tool ensures coherence among AIT activities and avoids redundancies or gaps.

During the phase 0+A, the mission definition phase, V&V shall show that the mission concepts and requirements are realistic and feasible and the requirements and specifications represent the real mission needs.

In the Phase B and C, the design phase, V&V answers to questions as: will the design be feasible which fulfils the specifications? Will the system be able to complete the mission and provide the required performance under real world conditions?

During the phase D, until the qualification, V&V contributes to guarantee that the design functions and performances satisfy the specification and hardware and software implementation fulfill the function and performance requirements for the mission under real world conditions. From the manufacturing point of view, V&V helps to verify that the flight items ‘as built’ in all aspects, i.e. physical, function and performance, fully correspond to the ones which passed through the qualification phase and all elements are properly integrated.

Moreover, it is important to investigate all those unknown or completely covered by the requirements effects in real world that could potentially cause a risk in the operational phase.

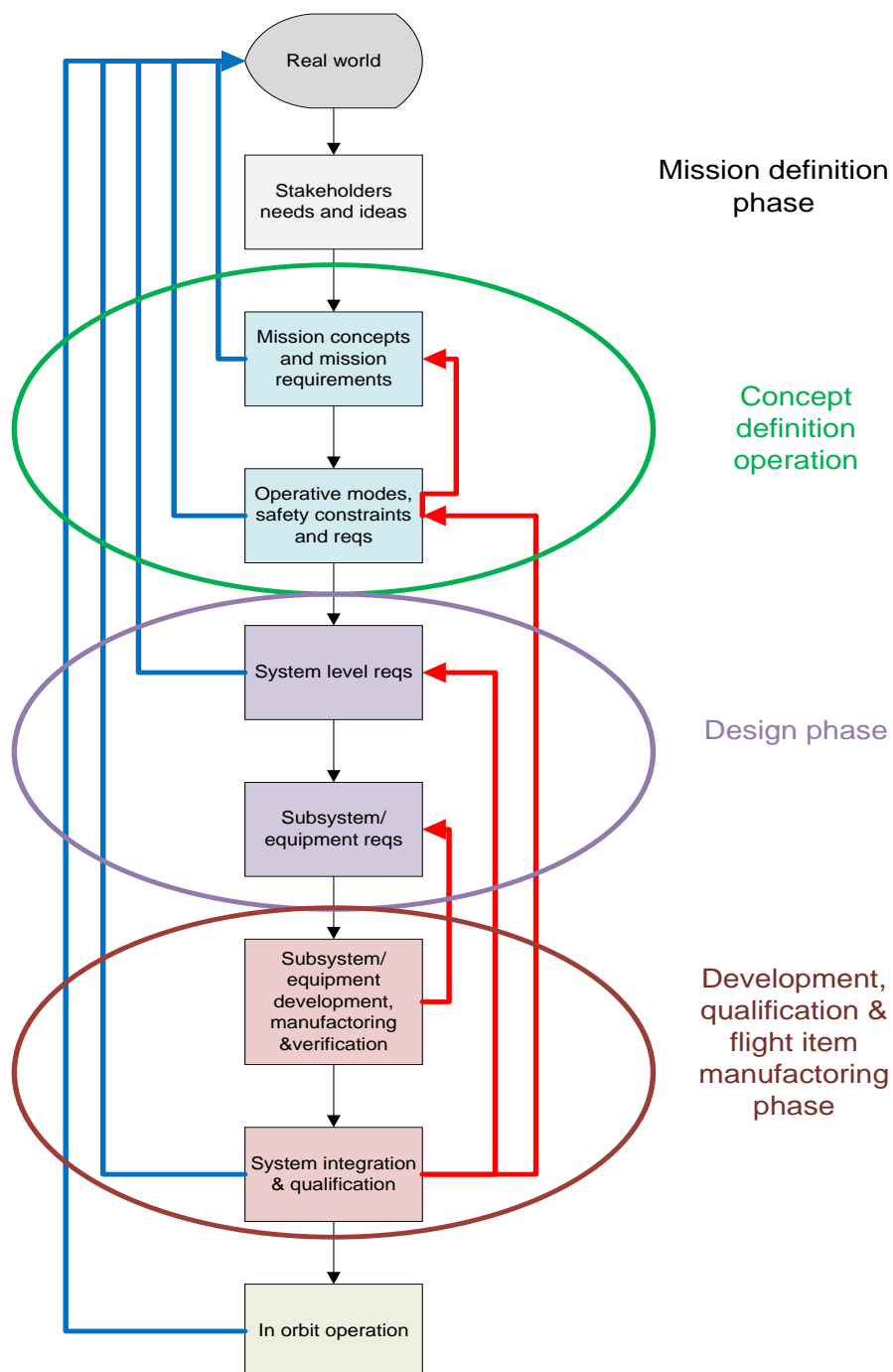


Figure 5: Verification and validation process in the space product life cycle

In Figure 5, the verification and validation process through the product life cycle phases is highlighted: the red lines represent the verification tasks and the blue lines represent the questions concerning validation.

Verification tasks can begin only once the first products have been established according to a set of written requirements or specifications.

The first task is usually the verification of the system specification w.r.t. the overall mission requirements. However, a major validation task already exists right at the beginning of a project: the stakeholders will have an idea of the mission to be performed. This first idea consists also of the ‘real world’. Starting from this idea, mission concepts and the mission requirements are defined and shall be validated against the ‘real world’, which at that stage is known only to a limited extent.

The “real world” shall be investigated and described along the entire development life-cycle of the project: this is a main purpose as well as the development of the spacecraft systems, subsystems and items.

After the first definition of mission and system requirements, all subsequent lower level specifications and implementations have to be verified against the next higher level specifications and requirements. However, as all these sets of requirements and specifications refer to the ‘real world’ describing the future mission, there will also at each step be the need to validate the detailed description of the ‘real world’ required for the verification task of that particular step. The verification/validation effort culminates in the development phase, where it should be proven that a system or item will achieve its specified performance under all conditions of the real mission. The sum of all these activities is referred to as *qualification*. However, keeping in mind the limitations of all verification and validation efforts, it will eventually be only in the operational phase of the real mission that the spacecraft system will meet the ‘real world’ proper. Only then can final proof be achieved that the system indeed functions and performs properly, that all requirements were indeed defined correctly, comprehensively and in sufficient detail, and that the tools for verification have indeed been correctly validated to represent the ‘real world’ to the necessary extent.

For the verification the effect of the environment shall be taken into account only if the environment can change the parameters involved in the dynamic process of the system i.e. GNC.

1.2.2 Verification strategies

There are several verification strategies which could be exploited in a complementary way. The present work should be compliant with the ECSS. [6] explains the following type of verification strategies:

- **Analysis.** A verification method which entails performing a theoretical or empirical evaluation by accepted analytical techniques. The selected techniques may typically include systematics, statistics, qualitative design analysis, modeling and computer simulation.
- **Inspection.** A verification method that determines conformance to requirements for constructional features, document and drawing conformance, workmanship and physical conditions without the use of special laboratory equipment, procedures or services.
- **Testing.** A verification method wherein requirements are verified by measurement of product performance and functions under various simulated environments.
- **Review of design.** A verification method using validation of previous records or evidence of validated design documents, when approved design reports, technical descriptions and engineering drawings unambiguously show that the requirement is met.

Another similar categorization is provided by NASA and DOD that define the following verification strategies [7]:

- **Analysis:** verification strategy elaborated from mathematical models, system analysis and simulations. These strategies may be complementary to tests, however they could replace

them when verification by test is impossible or unfeasible (i.e. mechanical essay of deployment, zero gravity ground deployment) or too much expensive. Simulations allow also to calibrate data and to treat errors

- **Similarity:** it addresses any equipment already exploited by further space missions or COTS. It has to prove that new system requirements and performances fit well within the already qualified component. Moreover, it has to report any difference from the reference product or from the planned tests.
- **Evaluation:** inspection (quality control), demonstration (i.e. interfaces coupling), project reviews (critical examination of technical documentation), in order to ensure material conformity to needs, redundancies, etc.
- **Testing:** functional tests in room and in space environmental conditions (functional tests under environmental constraints as vibration, thermal vacuum, EMC). Functional tests concern compatibility, interfaces, performance and operation which address electrical, mechanical, radio-electrical performance criteria. They are accomplished at each step of system lifecycle (qualification, flight review).

As reporting in [8], different kinds of tests for the verification can be performed. Going out from the specific categorization, two main subsets of tests have been individuated: the “functional and operational” tests and the “environmental” tests.

- The functional shall verify the system functions and capabilities. Functional tests are performed under specified ambient conditions and in all the operative modes. A related class of tests are the performance tests that shall verify the system (and implicitly, the subsystems and components/equipment) performances according to the relative specifications. Often functional and performance tests are combined. The key parameters vary in their ranges and according the mission profile and in-flight operations. They also contain the interface requirements tests in which to verify the compatibility among components, sub-systems, modules, aggregate systems, launcher adapter and ground support equipment. Within the functional tests, the system performances are evaluated e fixed. Life tests can be considerate as particular functional tests: they shall show the ability of the system to withstand the maximum operating time and the maximum number of predicted operational cycles during the “product lifetime” by providing the required performance at the end of life. These are very stressful tests that generally are performed on equipment by the relative provider. Electrical and RF tests are also performed. More detailed discussion about this kind of tests in proposed in the simulation strategies and configuration paragraphs and, in general, along the whole thesis.
- Within the environmental tests category the following groups of test are considered:
 - Mechanical tests,
 - Physical properties measurements
 - Model survey
 - Static load
 - Spin
 - Transient
 - Acoustic
 - Random vibration
 - Sinusoidal vibration
 - Shock
 - Structural tests
 - Proof pressure
 - Pressure cycling
 - Design burst pressure

- Leak
- Thermal test
 - Thermal vacuum
 - Thermal ambient
 - Thermal balance
- EMC test

In general lifecycle environment could be ground, launch and operational environment. Since a space system development requires time, space components may stay sometime in a ground environment before launch. Therefore, system engineers have to take into account ground environmental constraints: ambience conditions (air, oxygen, humidity and pollution), transport, maintenance, storage, safety, interface with ground support equipment and facilities. Interface requirements with launcher are specified by the launcher interface control document, along with mechanical (vibration) and system capabilities shall be verified in thermal and vacuum conditions. Space environmental conditions require ensuring that the space system is compliant with thermal, radiation, solar arrays and micro-vibrations constraints.

The V&V of functional and operational requirements by analysis and test are of interest for this thesis which aims at improve the entire V&V process to increase the reliability of and the confidence level in a space system reducing cost, time and resources.

1.3 Model and Simulation based approach

In the last decade a great support to system engineering methodology comes to the Model & Simulation (M&S) based techniques. Today, M&S provides a wonderful support for engineers and managers that must solve complex problems against reduced budgets, fewer resources and shorter schedules. M&S allows evaluating all the aspects related with the design, the development, the manufacturing and the operation of a space product because a great number of information and solutions can be provided and evaluated in order to make decisions.

M&S based approaches are found on “simulate-test-simulate build” sentence that is in opposite to the traditional approaches that refers to “design-build-test” sentence. These new approaches help to reduce the costs of a project because the real capabilities of a system, subsystem, device or equipment can be verified before its construction, saving time, reducing risks of failures, and shoot down the costs of their activities without the risk to downgrade the performances of the final product.

Moreover, the costs for hardware drop weekly while processing speed and storage capacity increase almost daily or hourly. Microchips containing more than a million circuits are able to performed advanced computation and manage real time process. From the software point of view, operating systems capabilities increase every day in order to take as much as possible advantages from the small and powerful hardware. First programming languages limits have been overcome from “all purposes” (capable to satisfy any kind of application, i.e. real time, operation on database, interfaces with user) languages such as JAVA, Visual Basic, C++, UML which are more powerful and flexible. User-friendly operating systems with graphical interfaces have made it easy to handle hardware and communications, thus cutting the time required to develop input and output routines. The internet, distributed interactive simulation network, and higher-level architecture are breaking down barriers to acquiring knowledge and significantly improving the ability to share information, tools, and methods.

M&S should be of interest not only for the technical aspects but also for the management of a research or industrial program because simulation allows building potential solutions, both qualitative and quantitative, that avoid doubts or uncertainties about decisions and choices. In this way the project can focus on key points and effective operational envelopes: through the M&S the operative real world condition can be reproduced providing a prediction on how the system behave and detect possible causes of failures and investigate corrections.

Definitely, M&S should help to solve complex problems under constraints about shorter schedule and time, fewer resources (in terms of personnel and money): in particular M&S leads to easily investigating every aspect of a project in any phase (designing, developing, producing, verifying and operating the space product).

M&S shall have instruments and tools that as quickly as possible generate information to support solutions that limit program risks and carry forward the project in any program moment. For these reasons, historically, M&S has increased its field of application and actually M&S has passed from hypothesis of project development technique to “must” for a space project in every phase.

M&S technique should be compared with the traditional ones where the approaches foresee to spend a lot of money and time for prototyping and “train and error” every solution.

M&S complexity requires good skills of programming, schedule and management because it transcends the single discipline: in fact, it applies to many activities such as planning, analysis, testing, and operations. Coordination between different entities that should provide models, perform simulations and analysis results is a key point for M&S methods: latency time and wrong communications can often be ingredients to which pay attention. Simulations should be flexible enough to use and reuse models across a system’s life cycle from development through fast-paced technology modifications and upgrades.

1.3.1 M&S based approach: advantages and disadvantages

Advantages and benefits derive from the application of M&S based approach.

- M&S provides the ability to test concepts and evaluate strategies before their application without the usage of hardware reducing costs and development time: in fact, potential solutions can be quickly investigated and compared and bring rapidly to conclusions without expecting the on board hardware and software availability. This fact allows extending the M&S methods to the first phases of the project as well as the development before the production. In these phases the laboratory (or the operation environment) is concentrated on the “computer” reducing risks and increasing (proportionally with the computer speed) the number of investigable hypotheses.
- the capability to quickly obtain information and data to analyze and compare provokes and encourages ideas, also strange ideas.
- through the simulation, key parameters about performances and costs that should allow to solve a problem can be indentified and evaluated: from a different simulation runs, the trend and the changes of values related to characteristic system parameters (i.e state variables) can be monitored in order to verify or negotiate requirements or review parts of the previous choices.
- Simulation carries out to predict behaviors of a system, evaluate results, individuate operation limits and critical aspect, and eliminate wrong solutions before the implementation (software) or the production (hardware).
- Simulation can also educate the system user showing the system capabilities and train the operators: the product becomes more user-friendly. A better understanding of the system behavior can be provided to the users and the operators that will reach an increased readiness.

M&S presents also limitations, disadvantages and critical issues:

- Before starting the simulation activities, assumptions shall be imposed and the risk is to leave uncovered criticalities: wrong assumptions lead to wrong solutions and conclusions. For this reason, it is fundamental paying attention to understand and identify assumptions and avoid forgetting or representing in wrong way critical points.

- Simulations could provide few or too data: in the first case, the simulation run results useless; in the second case a great effort to trace all the needed information is required. If the number of parameters to take under control is too high it is difficult to manage them and monitor their evolution and precious information can be lost and criticalities remain hidden.
- Difficulty or impossibility to model all the phenomena, devices features and relationship, provoking an incomplete explanation of the system. It results that final verification and validation shall be made with certified and standardized M&S tools or using directly real HW and SW.
- The real world remains in any case simplified and to determine how well a model shall represent a phenomenon or a device is not trivial.
- At the beginning, M&S approach requires time, money and human resources for the design, development and validation so an effective planning and a rigid control are needed to produce tools on time and within the budget.
- Too complex tasks (i.e. complex models) and excessive simulation constraints or requirements) can eat up too time and resources and kill all the benefits of M&S approach. Complexity of the models and simulation architecture shall remain sufficiently low to avoid a resource wasting but sufficiently high to well represent the real world: this is the focal critical trade-off that the M&S tools designer must take into account.

1.3.1.1 *M&S based approach maturity in space field*

[9] talks about the M&S based for a Space System and in its last release of April 2010 also tries to evaluate the maturity of this approach. ESA defines it as “innovative” and very powerful but at the same time it actually is premature to provide a full definition or, even, standardization. However it pushes organizations to progress toward a higher level of maturity in virtual engineering and design, they adopting an increased number of elements of the virtual and digital design approach and the development of facilities and tools able to verify the designed system.

1.3.2 **Computers importance in the M&S based approach**

The computer and, more in detail, the simulator is a key element in the M&S: computers are employed as virtual and artificial “laboratories” to describe, investigate, and understand a system’s behavior. The mathematical and stochastic characteristics of a system become computer software that is able to capture scenarios and conditions, producing valuable information to support a broad range of decisions for development and operation.

In a simulation based approach, computer simulation applies throughout the development, not just for analysis but as a development tool: in fact, thanks to the simulation, it is directly possible to analyze requirements, evaluate and allocate functions or objects, complete preliminary and detailed designs, synthesize code, integrate components or subsystems, and test .The design, development, integration, and verification process is continuous if a M&S based approach is followed. Moreover, the simulation helps the performances validation, the deficiencies avoidances and/or correction, and the system evolution throughout the life cycle phases and in any direction.

Summarizing, computers can be used:

- To simulate a real element, creating an high number of possible scenarios characterized by different conditions and deriving from specific or general assumption and, then, evaluating the behavior varying the main parameters.
- To investigate the systems capabilities modeling a real phenomenon. At the end of the simulation, data are compared with the real case: more the difference is small and more the model represents the real world

- To apply the inductive method for the description of a phenomenon: it consists in the simulation and then the verification of the mathematical bases of hypotheses.
- To discover scientific laws: simulation based experiments with system belonging to the same kind allow discovering or validating properties in relation with previous made hypotheses.
- To validate a methodology: performing a sufficiently high number of well documented positive simulation sessions allows validating a methodology.

1.3.3 Use of the simulation along the product life cycle

System Engineering process starts from the top level requirements analysis which flows into functional analysis and allocation. Tailored to M&S approach, inputs and outputs become simulation parameters that are strictly connected with hardware and software. The main idea is “model-simulate-fix-test-iterate” that help to identify deficiency in hardware and software bugs. Functional analysis helps to determine what the system can do and it ends with the allocation of the lowest level functions to the elements that conduct those functions. The elements can be subsystems, assemblies or components; it depends on the current iteration of the process. Architecture and interface among elements result in blocks diagrams or similar tools. From the previous work, designers can trace functional and performance, interface and design requirements. System requirements should be sufficiently detailed to provide design and, also, verification criteria. To simulate system functions and elements the models substitute the relative blocks and relationships in the defined architecture; engineering and physical parameters characterize each model. Hardware devices, software parts and interface can progressively replace models with the simulation loop as well as the system is mature, eventually resulting in SIL, CIL, HIL (see paragraph 1.4). In the alternate iteration of design and verification processes, the final physical architecture comes in the light and the simulation sessions on it bring to achievement of the functional and performances requirements. For modeling and simulation, the architecture shall always be sufficiently detailed to verify that the design meets the requirements.

Summarizing, the support of the M&S based techniques as part of SE consists of two “loop”:

- The design loop in which an iterative comparison between the evolving design and the system function occurs for ensure the goodness of the design wrt each function. Using simulation, it is continuously possible to monitor that the design meets the requirements
- The verification loop in which tests and other methods ensures that the design meets the operational needs according to the requirements analysis. Again, simulation helps to track and check the key parameters and monitor their evolution according to the expected ones.

Table 1 shows the use of the simulation along the product life cycle. [5]. The use of system simulation evolves through system level requirements definition, analysis and design trade-offs, then to AIT at subsystem and system level, and finally to training and support for operations. Simulation should be used as key element to support a wide range of engineering and operational activities during the lifecycle of a program. These activities are:

- Analysis, definition and validation of system and technical requirements.
- Validation that the design (from an electrical, thermal, mechanical, operational, etc. point of view) fulfils the high-level (mission and system) performance requirements.
- Software verification and validation.
- Development of GSE and test procedures.
- Prediction of system performance.
- Control centre and crew operator training.
- Operations procedure development and validation.
- System (failures and anomalies) troubleshooting.

	Product Life Cycle						
Engineering and Operations activities	Phase 0	Phase A	Phase B	Phase C	Phase D	Phase E	Phase F
Feasibility and Performance Analysis/Trade-Offs	Concurrent Design Activities						
Requirements Specification	Concurrent Design Activities	System & Mission Analysis					
Design Verification		System Interfaces and End-to-End Design Trade-off					
System and Mission performance verification		System Interfaces and End-to-End Design Trade-off					
Functional (Subsystem) V&V			Interfaces and End-to-End	AIV OBSW	AIV OBSW		
Spacecraft Qualification & Acceptance				Virtual AIV	AIV SVTs		
Ground Segment Qualification & Acceptance				MCS Testing	Operations Procedure Validation SVTs		
System Qualification & Acceptance					AIV SVTs	System Maintenance (e.g. Software)	
Training & Operations					Mission Control Team Training	On-Going Mission Control Team Training OBSW Patch and Ops Procedure Validation	Investigation of Disposal options

Table 1: System simulation in Product Life Cycle [5]

To develop a simulation successfully, designer must clearly describe its intended use (what?, why? and how?). Figure 6 shows the general uses of simulation to support SE. From this figure, it is clear how the use of simulation changes over a traditional program's life cycle. In applications, you can repeat each use for a spiral or incremental development.

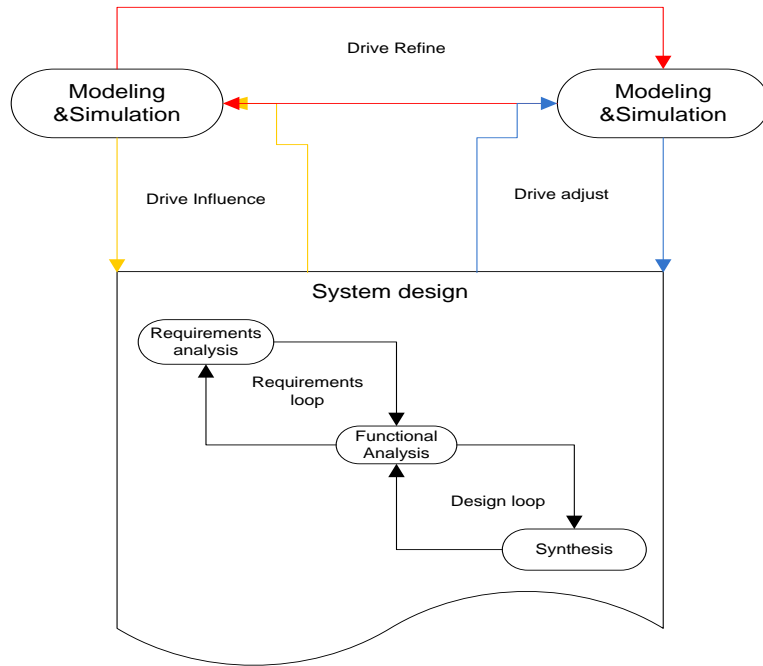


Figure 6: System development based on simulation

For simulation-based development, simulation is used in each development step to represent the entire system and, if required, each component. In either case, the models represent the system's components, and you can replace these models with software or hardware in the loop (SIL or HIL).

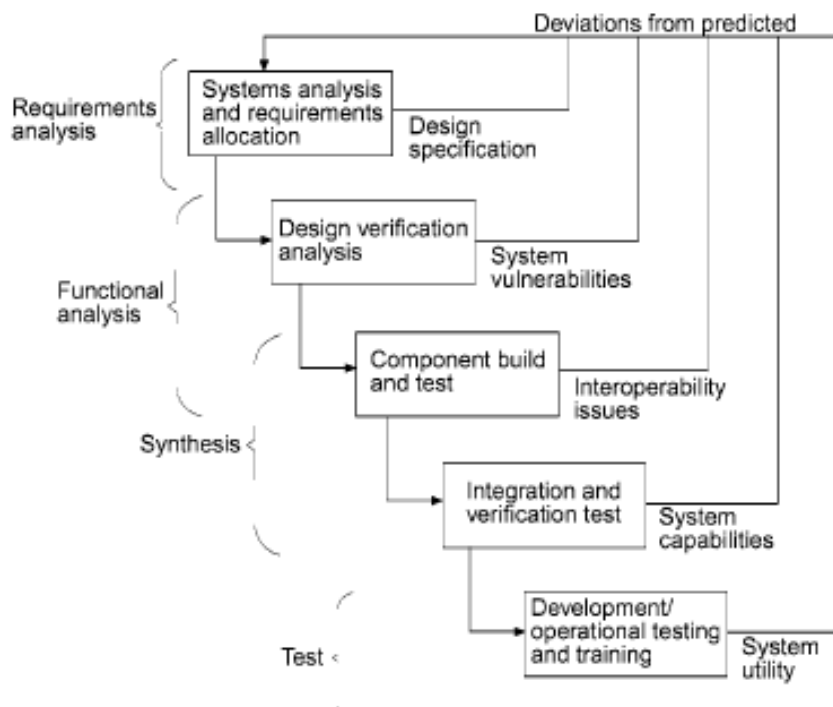


Figure 7: Use of the simulation in the product life cycle [7]

As shown in Figure 7, simulation based development can be split in stages:

1. System analysis and requirements allocation: simulation helps to predict the performances of concepts and designs. It is the basis for operational analyses that allow understanding the capability of a potential system. These can be simulations on specific part or function or on

the whole system. Analysis from this type of simulations provides a fully traceable component requirements and detailed estimates of performance. A lot of simulation runs are needed requiring that each run is as fast as possible independently by the level of detail of system models as well as the environment models. The simulation should output values for technical performance measures (key parameters) from evaluations of each design. The result is a clear statement of a component's expected contribution to these key parameters. Simulator is flexible enough in order to change easily the configurations.

2. Design verification: it means to define and trace functions or parts, allocate tasks, evaluate alternatives, in order to extract the preferred configuration. The simulation serves to demonstrate that the design fit the requirements. Simulation-based development uses an incremental approach to development supported by component models, prototypes, hardware and software deliveries, and simulations with hardware and software in the loop. Simulation verifies the design to ensure the components work as expected, are integrated correctly, and meet system requirements. Traditionally, designs use margins to handle large, complex programs that can't be tested in their full operational configuration without a lot of expense and risk. Space programs are an example. Often, measuring a system's design margins is difficult, as is tracking key parameters progress and the effect on system-level risk. In this case, your design may be too conservative (costly). To address these issues with simulation, the designer evaluates the key parameters for systems and components in each scenario or varies the key parameters within a component and observe the impact on its performance and on the performance of other system's components. High integrated tools are fundamentals to simulate system with different components and different level of abstraction and detail. Specific techniques allow comparing the results to other representations of the system.
3. Build and component test: it means to define the subsystems and their components and progressively testing ready software parts and hardware devices. System analysis and requirements allocation establish requirements from the top down. Component build and test predicts performance from the bottom up by estimating a component's performance as it integrates into the system. It can also handle new situations, such as a new environmental condition, altered component design, or operation at the corner of the performance envelope. This performance must be traceable to the system requirements. During preliminary design, designer must analyze different alternatives using detailed component simulations and input the characteristics of promising alternatives into tools devoted to evaluate the architecture. Without an integrated tool, this process is slow and laborious. Finally, the two processes shall be combined first using detailed component models with a fixed scenario and then incorporating the performance of component models into a simulator for system analysis and requirements. To do so, high detailed simulation components (or models) shall be integrated into a system-level simulation (integrated simulation environment).
4. Integration and verification: it consists of the integration of the whole system. Simulation provides the environment but differs in quantity. Build and component test (above) requires more because the latter stimulates and accepts responses from very low-level components. As more components become available, simulation decreases because real hardware and software replace it. During integration the system is the test object put in the simulation loop to ensure the items are integrated correctly, work properly, and meet the system's operational requirements. Integration and verification test follows this process by using components shown to meet component requirements, which enables the system to meet system requirements. The simulation results serve now to measure the performance of a system and its components against operational needs: pay attention that, in this stage, it is not of interest evaluate if the components meet performance requirements but the simulation shall verify that

the system design meets operational requirements. All the interfaces shall be active and become parts of the simulation because each component is connected through the natural component interface with the part of a simulation that uses a system's true data to work with the component : summarizing, the simulator must work in real time to support hardware-in-the-loop. Moreover, if applicable, the simulation should be able to operate in a distributed manner so geographically disperse system components can work together. This means the simulation's execution must be able to meet two requirements at the same time: it must support simulations of components that are not involved in the test and it must interconnect components separated from the local system simulation. Therefore, integration and verification test could require distributed, real-time simulations to integrate geographically dispersed components or subsystems (distributed executive).

5. Development and Operational Testing and Training: this stage foresees test in the real world; simulation is used to stimulate articles for test, connect test players, and provide the external environment for a test object. Simulation also helps to plan tests, make decisions and transform the results in settable parameters for the operations. Three abilities are required for the test: planning test, providing real time support during the test and training.

- **Test planning and analysis.** Support for testing includes planning for a test, predicting the test before it starts, and interpreting results to determine what happened and to relate results to the operational system's configuration and environment. If system components are not available, simulation replaces them. Simulation also provides or supplements the test environment. Test planning involves running simulations to determine component requirements that will meet a testing objective. It defines the best configuration for the test, including the component surrogates. The first output of the analysis is a set of go/no-go decisions to ensure the meeting of test objectives. The second output extrapolates the test system's performance to an operational system's likely performance by interpreting the results in terms of the design. Simulation-based development predicts the results before a test. If the test meets predictions, simulation has accurately captured system performance. If the predictions are wrong, the test may have occurred outside nominal conditions or a review to update the understanding of component performance in the simulation and, probably, in the design database is needed. Test planning and analysis require simulating the test conditions and articles in the test environment to reduce risk of failure. Therefore, the simulation must be able to model the test environment (models of test equipment). Models of the test environment and the systems shall consistently represent the real world, to relate simulated or test behavior to the operational system's behavior. Most tests will use GSE elements that produce behavior similar but not identical to the designed components. Simulation also must be able to relate overall test results to operational performance (translate simulated to actual system performance).
- **Test support.** Test support includes full, formal, developmental, and operational tests of the system, with extensive customer participation. In test planning and analysis, integrated distributed simulations run to verify the test can succeed and confirm conditions under which the test should be aborted or successfully completed. The test itself uses this data and real-time simulation to support decisions that control it. The simulation handles four activities:
 - Comparing real-time inputs on the test with nominal values
 - Simulating inputs from external systems or components that don't have surrogates available
 - Controlling the test environment and data systems
 - Collecting and recording data

Test support requires distributed, real-time simulations to help test rehearsals in nominal or off nominal conditions, to verify the test system's performance, and to support the real-time testing and data collection (distributed real-time executive). To validate test equipment, simulation must be able to connect to the equipment, inject test signals and simulated data, accept outputs, and compare outputs to results from simulating the system in a similar condition (test equipment interface and control). For test rehearsals, it must connect to each piece of test-control equipment and simulate in real time all normal and abnormal test conditions. It also should be able to pause, allow rollback and restart, and automatically collect data for comparison to other rehearsals and test predictions (scenario control). Finally, simulation must capture real-time data; process that data quickly; compare it to the outputs of simulation models, previous runs, or precomputed limits; and present the results (data capture and analysis). Of course, it also must be able to create a wide variety of scenarios (scenario generation).

- **Training activities.** To get good results, the operator shall be skilled. For this reason the operator shall be train people for operational testing and for follow-on operations. A training simulation must be able to clearly explain system options to decision makers and quickly help new people understand the system, as well as issues such as system requirements. Simulation should contain scenarios to illustrate other, more complex aspects of the operation, probably in a library of situations that a person can recall and run-or display if using a postprocessor (selectable scenarios). The user should control the scenario's speed by pausing, reversing, and altering selected views (scenario control). Other scenarios should illustrate key aspects of the operation to explain design decisions, system performance, and similar issues. For critical explanations, flexible scenarios allow the users to change them and then run a simulation based on the modified scenario (flexible scenarios). Moreover, the user's console controls the simulation, so the latter can work with the operational software's exercise and training mode (scenario control). Finally, the simulation must include in the report the ability to analyze performance (feedback) and provide tutoring to improve it-possibly through hyperlinks from the report (tutoring).

1.3.4 Methods, technique and tools

This paragraph reports the state of the art for commercial software for System Engineering, tools and facilities for software and hardware verification in space program, and provides an overview of the commercial software and tools for design and verification activities with respect to the life cycle phases.

1.3.4.1 *Tools for the complete SE process [10]*

In the last years, many software and SE modeling languages have emerged in response to the continuous advancements in the verification and validation field. These languages, created for an abstract, high-level description of a design and the components thereof, allow the designers to successfully cope with increasing complexities. Systems engineers have been using different documentation approaches to capture systems requirements and also various modeling techniques to express the complete design. Unfortunately, this diversity of techniques and approaches limited both cooperative work and information exchange. In order to ensure worldwide SE technologies compatibility and interoperability, international standards are needed.

Hence, various international standardization bodies are involved in SE, providing standard frameworks and modeling languages for SE. The Object Management Group (OMG) [11], the

International Council On Systems Engineering (INCOSE) [12], and the International Standard Organisation (ISO) are the main pertinent standardizing organizations.

Among standards, the Unified Modeling Language (UML) and the Model Driven Architecture (MDA) [13] are the most relevant and provide capabilities such as powerful visual design, execution, maintenance of software and other processes. Moreover, OMG has developed UML profiles, which are specializations of UML designed to support specific domains based on built-in extension mechanisms.

MDA is an OMG architectural framework and standard whose goal is to “lead the industry towards interoperable, reusable, portable software components and data models based on standard models”. MDA stemmed from the concept of separating the specification of the operation of a given system from the details related to the way the system uses the underlying capabilities of its platform [13]. In essence, MDA’s approach is to use models in software development, which includes both the specifications and the actual development of applications, independently of the platform supporting them. In the context of SE, model-based systems engineering (MBSE) is defined by INCOSE as “the formalized application of modeling to support system requirements, design, analysis, verification and validation.” It starts during the conceptual design phase and continues throughout the development and later life cycle phases [14].

1.3.4.1.1 Software in SE for M&S based approach

Modeling languages are commonly used to specify, visualize, store, document, and exchange design models. They are domain-specific, inherently containing all the syntactic, semantic, and presentation information regarding a given application domain. Various modeling languages have been defined by both organizations and companies in order to target different domains such as web development [15], telecommunications [16], hardware [17], software, and, most recently, systems (UML) [18]. Although SE has been in existence for more than five decades, up until recently, there has been no dedicated modeling language for this discipline [19]. Traditionally, systems engineers have relied heavily on documentation to express systems requirements and, in the absence of a specific standard language, have had to use various modeling languages in order to express a complete design solution.

This diversity of techniques and approaches has limited cooperative work and information exchange. Among existing modeling languages that have been used by system engineers, HDL can be cited [20]. In order to provide a remedy, OMG and INCOSE, along with a number of experts from the SE field, have been collaborating toward building a standard modeling language for SE. UML, being the modeling language *par excellence* for software engineering, was the language of choice destined for customization with regard to systems engineers needs. However, UML 1.x was found to be inadequate for such a use [21] and so the evolving revision of UML (i.e., UML 2.0) was issued, with features of special interest to systems engineers. In April 2006, a proposal for a standard modeling language for systems modeling, namely SysML, was submitted to the OMG, with the goal of achieving a final standardization process.

1.3.4.2 M&S based tools and facilities

1.3.4.2.1 TASTE [22]

The *ASSERT (Automated proof-based System and Software Engineering for Real Time system) Set of Tools for Engineering (TASTE)* is a framework of programs that aims at automating safety-critical software development. It is composed by a development consortium led by the European Space Agency (ESA).

TASTE program refers on SW dominant systems (but expandable to hardware in a near future) that in many cases is based on paper work, with few models, and with Microsoft Office as main tool. The

suppliers face difficulties to master design before testing and customers find review inefficient. In this context, the needs are to capture system model with the related properties, to verify early and continuously the specification during the design, to handle the heterogeneity of the system and to use automatic processes (i.e the code generation).

Key point in TASTE [45] is the definition of the system in terms of architecture, behavior, data, real time issues and hardware platform. From that, TASTE tools are able to generate code both for simulation and embedded system through famous software environment and language (i.e. Simulink, C++, SDL, ADA) - Figure 8.

The purpose of TASTE is to build Real-Time Embedded (RTE) systems that are correct by construction: based on high-level models, the toolset automatically configures and deploys complex RTE systems. To do so, TASTE relies on key standards and technologies such as:

- The Ravenscar Computational Model. The computational model enforced by the generated system, stemming from the Ravenscar profile.
- ASN.1 and AADL for systems modeling.
- Code/Model skeleton generators targeting major programming languages (C, C++, Ada) and modeling tools (SCADE, Simulink...) to help in designing systems' functionalities.
- Code generators to automatically generate systems, based on high-level models.
- PolyORB-HI. A middleware to map the primitives of generated codes to the ones offered by targeted operating systems.

At the beginning of this study, TASTE-generated systems were software only, and were targeting multiple platforms: x86 (with Linux, Mac OS X, FreeBSD, RTEMS), ARM (with RTEMS, Linux), SPARC/LEON (with RTEMS or OpenRavenscar).

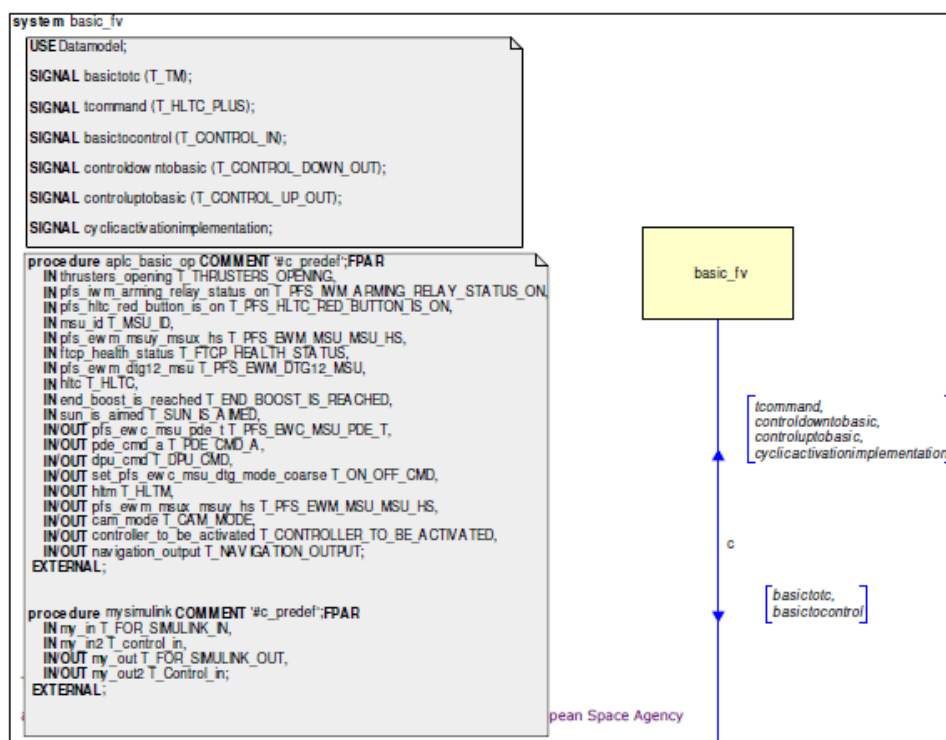


Figure 8: Example of the code generation made by TASTE

1.3.4.2.2 SIMSTB

SimSTB [23] is being designed and developed in the frame of future spacecraft programs based on ERC32, a radiation tolerant processors family used in space field.

The hardware architecture of SIMSTB is a standard distributed architecture. One Test Control Server is used for Environment administration, Configuration management, import and export of data with the system database and the SDE. Several test control workstations allow performing independently test preparation, monitoring and control of test execution and post processing of archived data. Several Simulation Processors that run all simulations are connected to the network. Both Test Control Server and Workstations run on UNIX platforms while the Simulation Processor is based on PC LINUX.

The software architecture of the test facility is derived from the simulator architecture. This one is based on Astrium Monitoring and Control infrastructure SIMGO, simulation infrastructure SiMIX and on the Software emulator SIMERC32.

SimSTB main components related to OBSW validation are:

- A real-time ERC32 software emulator, so-called SIMERC32, implementing a wide range of debug functions (obviously no real-time when in debug mode) and an interface to gdb (gnu debugger).
- Symbolic access to OBSW as well in command language as in synoptic and in any analysis functions.
- Integration with GDB: the control can be given to GDB that provides its standard symbolic functions thanks to its SIMERC32 interface.

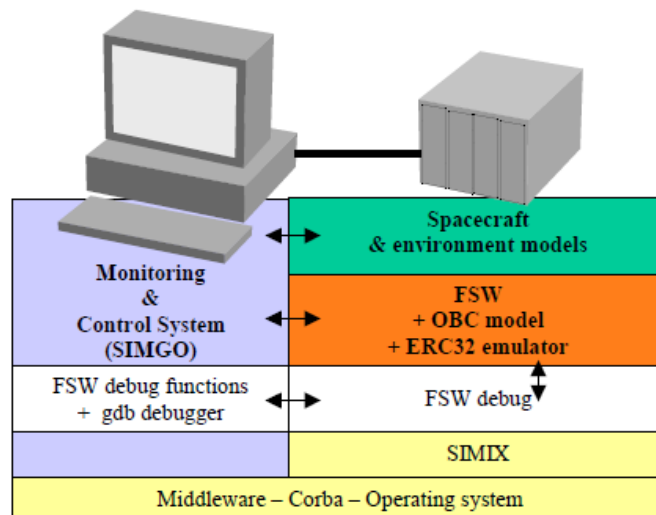


Figure 9: Typical SW simulation system architecture based on SIMWARE

SiMWARE is the result of an approach for building software components in a generic manner enabling to use these components from one test & simulation system to another and from one satellite to another. (Figure 9)

SiMWARE has been designed and developed as a set of reusable products meeting numerous objectives:

- **SIMIX:** A real-time simulation kernel providing all necessary services from the SVF needs (debug, archiving, real-time...) to the Satellite simulator need (multi-instantiation, failure injection, simulation state vector saving/restoring...) while offering classical scheduling functionalities. This generic kernel can be executed on different configuration (VxWorks, Windows NT, UNIX, LINUX...)
- **SIMGO/OC:** Specific Monitoring and Control services for operational satellite simulation and providing interfaces between SIMIX kernel and simulation components which can be complemented for System Test Bench Monitoring and Control or SVF facility. The main components involved in the validation facility are:

- LNG: for language and command procedures preparation, compilation and run-time execution. In that respect the command language supported by SiMWARE is that of Open Center. ELISA allows telecommand generation (depending on the On board software, TM/TC packets definition described in system database), to control simulation environment through access to telemetry, simulation and variables. For software validation purpose the operator can have access to flight software variables (read/write) in a symbolic manner, registers, set and remove break points and enable and disable traces.
- SYN: for dynamics synoptic preparation and real-time animation. The element provides monitoring components such that animated mimics, graphs, tables of Alpha Numeric Displays... to display raw or decommuted telemetry and simulation parameters.
- LGB: for activity (logbook)management for editing, displaying the execution status, on line and off line,
- ARC: for archiving busses traffic, simulation parameters and telemetry for further postprocessing,
- TRE: for tests report and evaluation to perform OBSW traces and test archive flight software coverage and performance analysis as well as error investigation and correlation between test data and environment simulation.

This approach allows building easily the Monitoring and Control function according to needs from the SVF, Table and System Test benches and the Operation simulator.

SIMWORK: a graphical development tool, producing automatically documentation and C++ code directly compatible of the simulation kernel SIMIX. This tool also allows encapsulating already existing model code in C/Fortran language to generate the necessary SIMIX interfaces, minimizing thus modification on already existing models SIMVAL: models test tool based on SIMIX and allowing models triggering either directly or through on-board bus (1553, OBDH) or spacecraft TC. SiMWARE has been integrated and is in use within about 20 spacecraft simulation systems, in conjunction with Open Centre. (Figure 10)

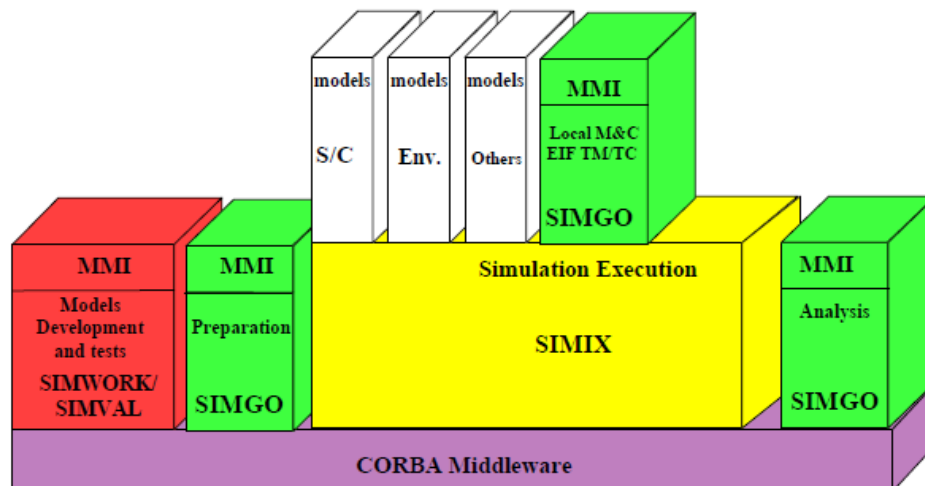


Figure 10: Logical architecture of SVF infrastructure and models

1.3.4.2.3 DLR-EPOS 2.0

“EPOS 2.0”[43] is the second generation of the “European Proximity Operations Simulator”, a facility serving Proximity Operations Simulation since the late 1980s at the German Aerospace Center site in Oberpfaffenhofen, Germany. The previous EPOS facility was a test bed developed by DLR for the simulation of spacecraft maneuvers notably over the last few critical meters of rendez-vous phase (prior to physical docking). The facility consists of a large mobile platform used to hold the RVD

interface. The platform can provide 6-DOF translational and rotational motion to the RVD hardware interface. The last intensive utilization of the facility was the test and verification of the ATV RDV sensors and systems which are used for approach ISS, e.g. the Japanese HTV.

Future applications for satellite on-orbit servicing missions require the EPOS facility to be able to provide the following test and simulation capabilities

- the 6-DOF relative dynamic motion of two satellites in the final approaching phase from 25 meters to 0 meters.
- the 6-DOF contact dynamic behavior during the entire docking process including the initial impact, soft docking, and hard docking (final rigidization).
- the space-representative lighting and background conditions

EPOS 2.0 uses two 6-DOF robotic arms to provide hardware-in-the-loop simulation capabilities for Rendezvous and Docking. "EPOS 2.0" offers extended motion ranges, faster computational speed and higher payload capacity than the old gantry.

The facility comprises a hardware-in-the-loop simulator for physical real-time simulations of rendezvous and docking maneuvers. This test bed will allow simulation of the last critical phase (separation ranging from 25m to 0m) of the approach process including the contact dynamics simulation of the docking process.

Moreover, its main advances are:

- It is a highly accurate test bed. The measurement and positioning performance will be increased by factor 10 compared to the former EPOS facility.
- Dynamical capabilities will allow for high commanding rates and the capability of force and torque measurements.
- The simulations of sunlight illumination conditions as well as the compensation of Earth-gravity force are both part of the assembly to generate an utmost realistic simulation of the real rendezvous and docking process.
- The utilization of standard industrial robotics H/W allows a very high flexibility related to different application scenarios.

The new facility consists of the following components:

- A rail system mounted on the floor to move an industrial robot up to a distance of 25m,
- A KUKA KR240 robot (robot 1) mounted at the end of the rail system for simulating the 6 degree of freedom of the second spacecraft.
- A KUKA KR100HA robot (robot 2) mounted on the rail system for simulating the 6 degree of freedom of one spacecraft.
- A PC-based monitoring and control system to monitor and control the RvD simulation on the facility. It can be

Figure 11 shows the assembly with two robotic arms, the one in the foreground fixed to the ground, the one in the back displaceable on a rail.



Figure 11: EPOS 2.0 facility

Table 2 gives the main features of the EPOS2.

Performance parameters	EPOS2
linear traversing range	25 m
maximum payload	100 kg / 240 kg
commanding frequency	250 Hz
positioning accuracy	0.5mm / 0.2°

Table 2: EPOS 2.0 main features

1.3.4.3 *Tools and facilities specific for the one or more life cycle phases*

This paragraph shows the features and use of tools and facilities for SE activities along the product life cycle.

In the feasibility phase, the tools shall be able:

- To perform orbit and trajectory/orbit simulations
- To manage budget analyses through linked spreadsheet
- To provide 2D/3D visualization
- To manage databases of data for analysis
- To generate report
- To manage databases

The facilities and infrastructures that perform such kind of operations stay at prime contractor or space agencies and their name varies accordingly: “CDF” in ESA, “Satellite Design Office” in Astrium, or “Project Design Center” at NASA/JPL. The analyses in this phase require few computations performed with general purpose tools or software (i.e. Office Excel) and no functional simulation is performed because the system is not sufficiently detailed and any control functionalities are still undefined from a quantitative point of view. Actually, with respect to the system, orbit propagation analysis is needed. Satellite Tool Kit [24] is the widely application software used for this purpose.

Traditionally, in phase B simulation tools are greatly applied in the system engineering process for overall analysis and subsystem design analysis. Besides tools for functional simulation, specific analysis is carried out in the disciplines of structure mechanics, thermal engineering and electric design using dedicated tools. These comprise for example, thermal analysis, (e.g. with the tools ESARAD / ESATAN / FHTS), structural mechanics, (e.g. applying ANSYS^s, NASTRAN^o), and other

fields. CAD tools are frequently used in this phase for qualitative analysis, i.e. CATIA and SolidWorks.

For functional simulation in phase B commercial toolboxes which comprise special libraries for control engineering and system dynamics engineering are used. Matlab together with the add-on toolboxes Simulink and Stateflow is the most popular and commercial software environment of this category. Matlab competitors are SciLab (open source) and Modelica. The Figure 12 shows the layer of a Matlab-Simulink satellite attitude control system, but similar simulations can be conducted for other engineering disciplines but mainly at subsystem or component level.

Any time that design process completes a step, verifications campaign should be done. As seen before, the verification ranges from the lowest equipment and piece of an algorithm up to the top level of the whole system. The possible scenarios have been described in paragraph: they are AIL, SIL, CIL, HIL. Tools and infrastructures for these purposes are quite uncommon and they are even developed within big industries and remain devoted to a specific program. An example is Model based Development and Verification Environment (as well as its successor FVI- Functional Verification Infrastructure), a satellite simulators environment self developed by Astrium GmbH applied in all the Astrium – Satellite sites. Figure 13 reports the complex architecture of MDVE [25], [26].

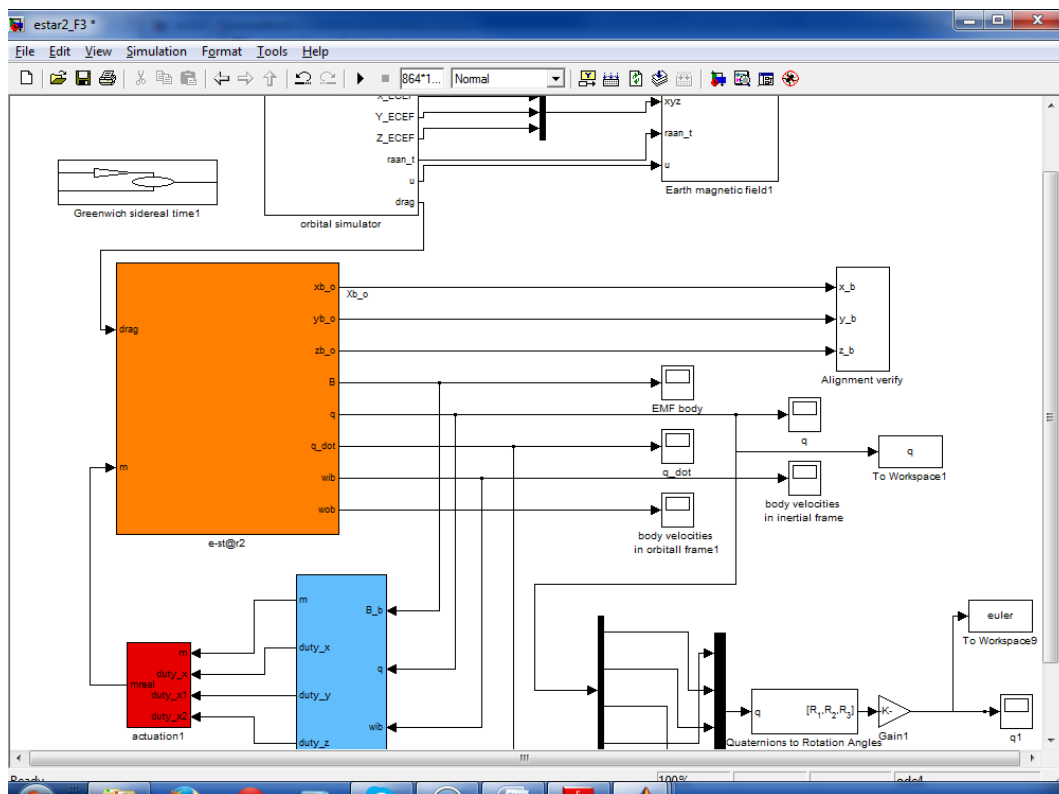


Figure 12: Matlab-Simulink ® example

The core of the MDVE infrastructure is the system simulator, called "Real-Time Simulator". A subassembly of the system simulator is the on-board computer simulator, "OBC simulator", which models the on-board computer of a satellite. The complete system is controlled by a control console, the so-called "Core EGSE".

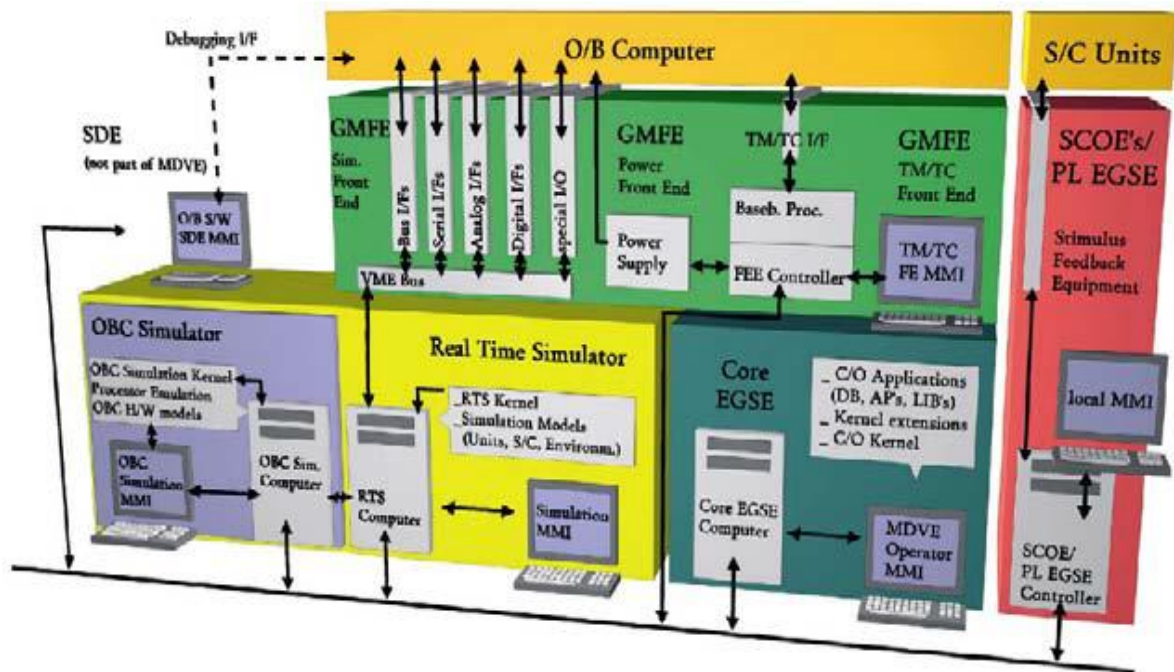


Figure 13: Model based development and verification environment (MDVE) - ASTRIUM

There is an interface between Real-Time Simulator and integrated satellite hardware, (e.g. real on-board computer instead of simulation), for hybrid bench configurations.

This interface performs the data transfer between integrated hardware and simulation, is responsible for the power supply of integrated hardware and for routing of telecommands and telemetry between integrated hardware in the loop, namely the on-board computer, and the control console. The simulator is modular and in each simulation session can be built adding/removing elements as a “LEGO” construction: this peculiarity allows various configurations for AIL, SIL, CIL and HIL. Specific Kernel are developed (generally in C/C++ language) and then standardized, in particular for guarantee the real time operations, unavoidable condition in particular in the final step of the verification campaign in which all the parts of the entire system must work together. Specific data base contain models that represent on board computers, sensor, actuators, etc... Each equipment is modeled as an occurrence in an object oriented structure.

The simulator is constituted by:

- A simple functional on board computer into which the tasks are embedded from Matlab Simulink to C++.
- Functional models of the equipment
- Numerical models of the space environment for evaluate the external influences on the simulated system

The numerical parameters are passed by the control console that manages both the simulator software and the on board algorithms. No Hardware is involved and, consequently, no careful is posed on the interface type or protocols. The final conversion of the model from Matlab/Simulink to the Object Oriented C++ is guaranteed by the Real Time Workshop tool (provided by Mathworks) that makes a rapid prototyping and derives the software structure, used in the next step.

The second step foresees the test and verification of the on board software. It means that the following elements are involved:

- The Operating System of the on board processor
- The system control code, especially the control algorithms
- All the interface drivers in input and output between the simulator and the on board processors

- The functions necessary for the command and telemetry rice-transmission

Much more models are involved with respect to the FVB and the real protocols are recalled to the data exchange, not only for the communication between simulator and system but also models of the sensors and actuators protocols could be applied. SIL could be performed using one PC in which the on board software runs and a PC (or Network of PCs) for the simulator. Focusing on Control Console, it provides the interface with human operators: for this reason it shall be user friendly and a graphical user interface, based on Java language, runs to support the initial setup and the final data management and visualization. At this step complex operation scenarios can be already reproduced and verified, i.e. close loop attitude/orbit control scenarios.

Third step foresees a simulation configuration suitable for CIL. On board SW shall run on the “real” processor in order to test all the functions, from elementary to the highly complex: a great difference with respect to SVF is the processor/controller that is the real one, instead the functional model which is built simply based on the controller documentation and datasheets from the provider. Different test subsets constitute this step. The first one shall allow the compatibility verification for HW (the controller) and the software (see figure vv). The success of this test leads to the out-and-out CIL whose architecture is shown in figure mm. The controller is definitely connected to the simulator that has to provide the still missing analog and digital signals and interfaces into the controller input: sometimes this can cause a problem of wires configurations. Other important points consists the real time: in fact, the simulator has to serve the controller in real time. It means all the interfaces and the protocols have to be processed in parallel which requires a sufficient computation power, a RTOS, and a real time capable data bus.

The fourth and final verification step leads to the HIL simulation, based on a so-called Electrical Functional Model whose architecture is presented in figure nn. It evolves from the STB by increasing the number of hardware components involved: each time hardware is ready for integration, it substitutes the respective model. For all that equipment not ready and for all that environmental conditions not reproducible the model remains in the simulation loop and the output is used as stimuli for the system. Often these stimuli are provided by Special Check-Out Equipment (SCOE) directly commanded by the simulator. Figure bb shows an arrangement for HIL of CryoSat1 in Astrium.



Figure 14: Astrium test bench [28]

Final integration tests are conducted at the entire system level. Now the physical design validation can start: it mainly means to execute thermal-vacuum tests, EMC tests, and structure and mechanics tests: they require ad hoc facilities, normally owned by space agencies or specialized technical services providers.

Finally, the simulator can be applied also during the operational phase (Phase E): In general, control console is replaced by the flight operations system installed in the spacecraft.

The resulting simulator setup in the ground station can be used for training of the spacecraft operations staff and for tests of software patches and bug fixes on the simulator before they are uplinked to the real spacecraft.

Pay attention that the "ESA Space Operations Center", (ESOC) typically does not accept system simulators from the development cycle because of their philosophy to use only tools for operations support which are independently developed. This approach minimizes the risk of potential inherent development process errors and such errors can be spotted during operation. For this reason the ESOC has developed its own system simulation infrastructure called SIMSAT [27].

1.4 Simulation strategies and configurations

Simulation is used as key element to support a wide range of engineering and operative activities during the space product lifecycle: Well-defined architectures having specific configurations are involved in the different phases.

The SE function is supported by a set of coherent and incremental test and simulation facilities from phase A up to phase E. The core of these simulation tools is a “virtual system model”, reflecting the functions and behavior of the complete system to be built at the level required to support the respective analysis and verification tasks. This virtual system model should be considered part of the overall model philosophy, and be explicit in the design, development and validation plan, reflecting its different configurations.

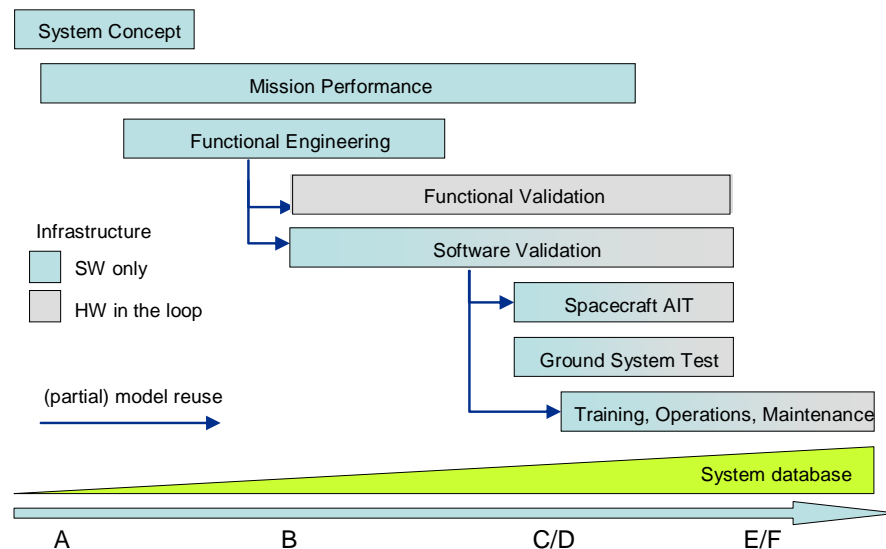


Figure 15: Tasks supported by M&S approach [8]

Each facility has a virtual system model that evolves with the increasing definition of the space system during each phase and from one phase to the next. On the base of the models that constitute each facility there are data-bases which are incrementally populated and validated across these various steps.

The virtual system model comprises the simulation infrastructure and the models of the space system, including the ground segment and space environment. It supports external interfaces to the GSE and the monitoring and control element of the virtual system model and related functions

Each facility thus represents a virtual instance of the space system or a subset of it. Therefore these facilities need to be integrated in the overall model philosophy at system level; covering virtual as well as physical models. The validation of the virtual system will be incremental over the life-cycle of the project, reflecting the status of the system (as specified, designed, built, tested...).

Figure 16 shows the general and more complete configuration of the simulator facility.

The development of simulator in space projects has the main task to support the engineering and operative activities, reducing the total cost. As seen, one issue is related to the modeling: types, parameterization, level of complexity, organization in database are central to allow the system simulation. The second point is to have generic and modular simulation architecture, able to provide different configurations in terms of fidelity/level of details of the virtual system wrt the real one, performances, development level/product life cycle. Developing “virtual/digital” models of a space system has a great importance just in the first phase of the project, evolving in the design phase, assembly, integration and verification phase and final operation phase. This process is coherent with the SW and HW development of the system: in fact, a simulator gives effectiveness to design and verification because:

One of the advantages that derives from the M&S based approach is the reuse of the models and items both from previous programs and from the previous phases of the same project, saving money and resources.

What is reusable?

- The firmware code used to configure the simulation;
- The simulator architecture;
- The parametric equipment, environmental, power, thermal, and dynamics models;
- The configuration data (specific for type of mission and system);
- The Ground Support Equipment.

The main enemy for the reuse is the incompatibility because different conventions occur and unimplemented protocols are required. In other cases, it may be more cost effective and lower risk to develop a new model rather than reengineer and validate an existing model. For this reason, a good simulator shall allow the development of new models providing adequate tools (e.g. a programming environment) but also specific roles for guarantee the compatibility with the existing architectures. Moreover, this feature is essential when the model of specific hardware shall be simulated or inserted in the loop, and custom features shall be modeled.

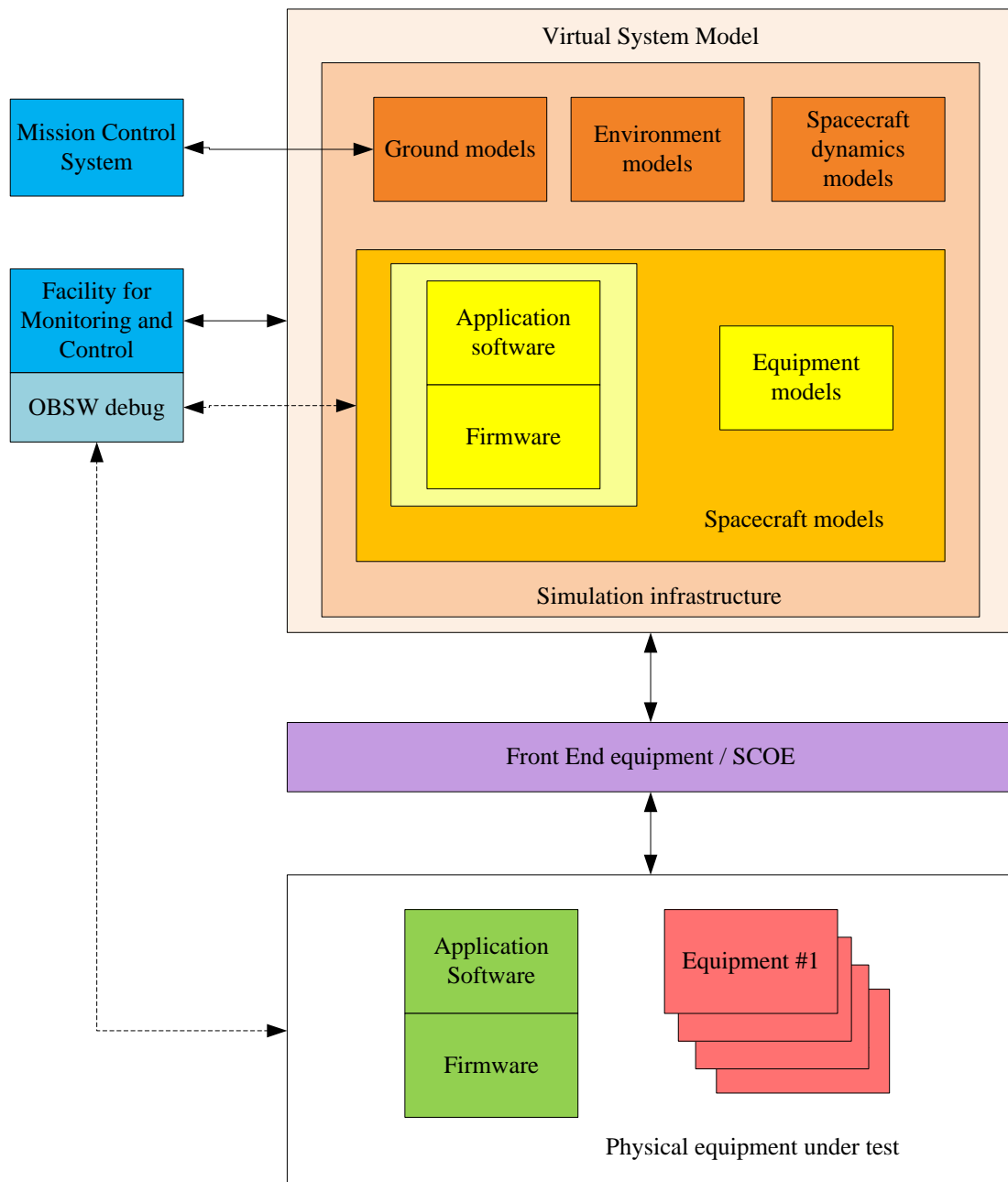


Figure 16: Simulation Facility Architecture Components

Equipment models (as well as the real elements) should be reused if within a new project the equipment are the same. However, some models remain specific for the project (e.g. the payload) or they are specific for the specific phase (e.g. the power model is more accurate in the development phase wrt the operational phase).

Simulation Models may execute the actual on-board software code (using an emulation of the processor instruction set and interfaces). Also, they may embed actual equipment (hardware in the loop).

Nevertheless, model compatibility exists and need to be considered for risk reduction / savings across or within projects:

- Generic models for equipment, subsystems or ground segment that are re-used across projects with or without limited modifications,
- Models fully or in part re-used across phases of a project to support diverse activities.

- Optimize the project in terms of requirements management and verification and design trade-offs:
 - Increase the coherency among analysis, project, verification and operations: assessment of boundaries and margins, system operability analysis.
 - Contribute to the reduction of number of project design process, their duration, the number of physical models involved, and the number and duration of the test activities: preparation and execution of test, model-based data sharing

A brief overview of different architectures and the possible related simulation configurations and executions is presented hereafter. In particular, traditionally three main categories are individuated:

- Analysis and design simulators
- AIV simulators
- Simulators for test, qualification and operations of ground segment and training

1.4.1 Analysis and design simulators

The analysis and design simulators are used for the analysis and the design, mainly for:

- Develop and integrate subsystems, e.g. GNC
- Individuate critical point for the whole system or the mission

1.4.1.1 System Concept Simulator

The System Concept Simulator (SCS) works in not-real time with low detailed mathematical models provided from the literature of the different disciplines involved in SE. It shall support a quick evolution of the concepts of operations and it shall be validated against the mission requirements in order to help the decision making and eventually re-negotiate the requirements. Simulation shall support the phase 0/A SE activities in terms of verifications of the functional design, design trade-offs decisions, and concept visualization. Analysis of potential concepts and trade-off investigation based on different alternatives on system (technical feasibility) and programmatic (schedule, cost and risk) is performed. The objective is to allow a highly iterative process that leads the tradeoffs (in particular among different disciplines) for system concepts by providing a quantitative assessment of the system performance for different mission and system concepts. The visualization of these concepts allows improving the awareness of the design and adds value to the presentation of the feasibility study results for customers or other partners.

Input	Output
First issue of the system functional specification	Consolidate mission and system concepts
Key parameters for each discipline involved in the design (GNC, power, mechanical, etc...)	Visualize alternatives
Generic system and environment models	Input for decision making
CAD/3D system models	Input to preliminary system functional specification
Operational scenarios	Input to tradeoffs analysis

Table 3: SCS input/output list

The main features of SCS are:

- It is constituted by a specific architecture (core and interfaces), system models, ground models, environmental and dynamics models, 3D models, command consol;
- It is only based on software (the simulator software!);
- Setup activities: to configure scenarios, to configure generic and specific mission models, to develop/adapt new models for mission specific details;
- Simulator validation: formal validation is not applied; generic models should be validated outside the setup process, as development part of the architecture;
- Reuse: maximize the reuse of the existing, generic models. They can come from previous project or from generic shared libraries. Historically, the reuse of models in this phase is up than the 80% [ESA sources].

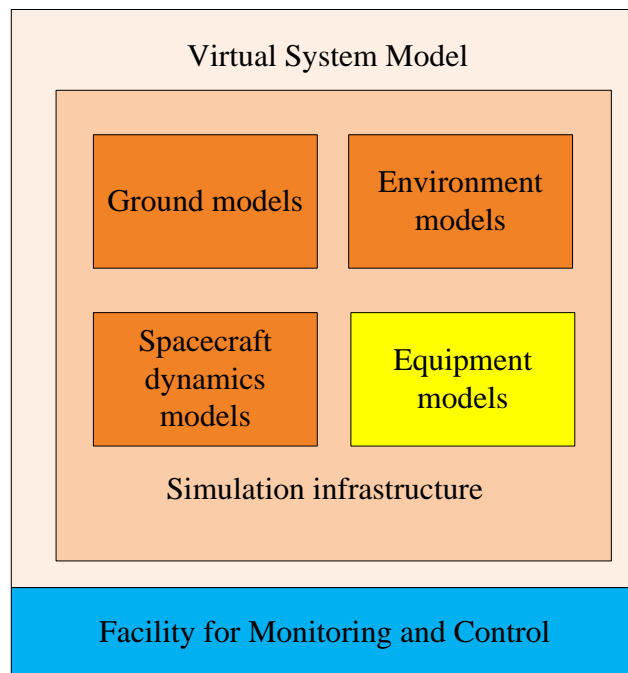


Figure 17: SCS - System Concept Simulator

1.4.1.2 Functional Engineering Simulator

The Functional Engineering Simulator (FES) allows the verification of critical system design elements (such as Data Handling, AOCS/GNC algorithms, and so on). It is recommended to maximize the reuse of the mathematical models (or parts of there) between these two to prepare the basis for building the real-time simulators that are exploited in the subsequent design/test phases. To this end, commonalities between the two classes of simulation models need to be considered. The FES contains all the functional models needed for algorithms validation, the functional organization of the real system, but not necessary representative of the real interfaces, of the data handling subsystem and of the protocols. The functional model is a model representative of the real modeled elements behavior. The main objective is to validate the system functional design; in particular: to support the system requirements consolidation, to validate the key algorithms needed in the subsystem (e.g. GNC), to verify system preliminary and detailed design, and to verify the system performance analysis the results of different simulation runs with specific setup (i.e. worst case, with perturbations, etc...)

The simulator shall reflect the architecture and the interfaces of the design and provide simulation capability to assess engineering requirements and algorithms performance. It should be easy

configurable and flexible in order to allow the introduction and configuration of the elements (also new elements).

Input	Output
System specification and design	Consolidated system requirements
Specification of subsystems required for the functional representation	Validated algorithms
Critical algorithms (e.g. attitude and orbit control and determination algorithms)	System performance assessment

Table 4: FES input/output

The main features of FES are:

- FES is constituted by the scenarios functions, the architecture configuration, the orbit, environment and dynamics models, the ground models (if needed), the subsystem functional models, command consol;
- It is only based on software (the simulator software!);
- *Setup activities*: models development, models and algorithms unit testing and validation in open-loop, scenario definition and setup, simulator verification and validation;
- *Simulator validation*: each model shall pass unit level test and validation in open loop by sending commanded inputs and verifying the proper answer and comparing the key parameters values wrt the and validation of known or expected behavior (e.g. orbit propagation results). The final validation is made with closed loop simulations against reference data of reference cases;
- *Reuse*: it consists of the reception and integration of components coming from open or closed loop engineering simulators developed along the project or deriving from previous projects.

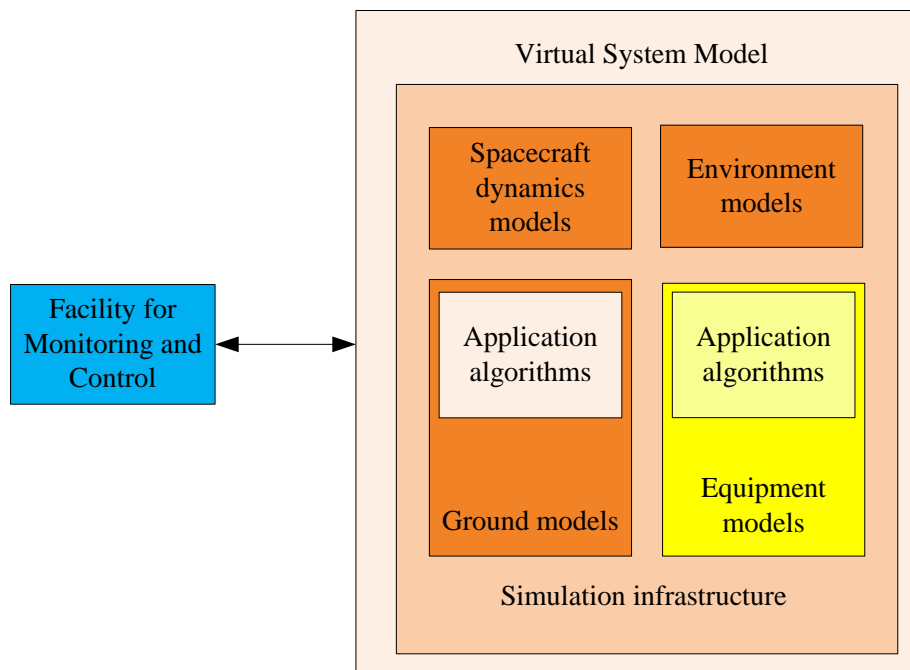


Figure 18: FES - Functional Engineering Simulator

1.4.1.3 Functional Validation Test Bench

The Functional Validation Test Bench (FVTB) is thought for analyze the system performances and finally validate the critical subsystem elements. The scope is to have a complete virtual system. Its objectives are to provide performance analysis, validation of critical elements/subsystems design, and test benches for hardware and software physical models. It should reflect the architecture and functional interfaces of the system design and should be able to perform functional simulations of the subsystem level. It is required to model the protocols and/or the electrical configuration of the equipments for the physical models needs.

Input	Output
Preliminary design data package with the identification of critical elements and their specification	Validated critical items design
Physical models	Input to decision making and technical specification
Criteria required for the assessment of suitability of the physical models for the system	Consolidated specification for the components

Table 5: FVTB input/output

The main features of FVTB are:

- It is constituted by a specific architecture, functional models, models for protocols and electrical interfaces, GSE for host and connect physical models, physical models (HW and SW) under test, command consol;
- There are two mainly configuration: only software (the simulation software) and HIL (EM);
- *Setup activities*: to develop, test and integrate the functional models and specific interfaces to element under test, to develop and adapt the environmental, dynamics, orbit models, to integrate the product under test, to develop the test plan;
- *Validation*: the functional model should be validated against the design specification, the simulator can only be validated with the product under test in the loop;
- *Reuse*: Orbit, environment and functional models of the FES and/or similar missions can be (partly) reused.

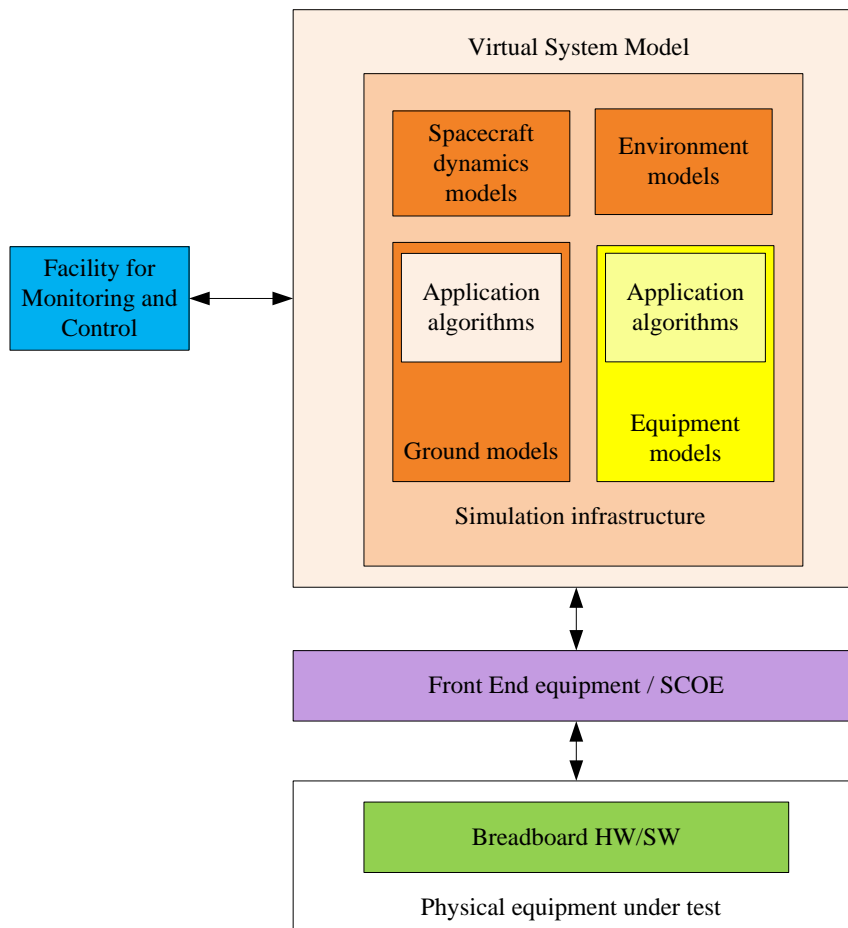


Figure 19: FVTB - Functional Validation Test-Bench

1.4.1.4 Software Validation Facility

The Software Validation Facility (SVF) has the task to support the OBSW validation, including Data Handling, AOCS and GNC and Payload software. This validation involves the lower layers of the OBSW interfaced to the OBC hardware (so-called firmware) as well as the upper layers (so-called application software) related to AOCS and GNC, data handling, mission management and control, monitoring and control of the payload equipment, thermal control and power control. One important feature is the ability to inject failures in the models that enables the user to trigger the OBSW monitoring processes as well as to exercise the FDIR mechanisms. Sometimes a less representative approach may be adopted - for example when validating the flight software against its specification: in this case, simpler so called “model responders” (or test stubs representing equipment) may be sufficient to test the open-loop behavior of the OBSW. Summarizing, the simulator shall perform OBSW integrated tests, parameter settings (e.g. GNC), functional validation in open loop, HW/SW interfaces verification in open loop, performance and robustness tests in closed loop, the validation activities of software maintenance, and the validation of the system database. The execution of the on board software shall be possible both in fully software simulated configuration and embedded into OBC HW. The debugging and (cross)-compiling capabilities are provided.

Input	Output
Equipment specifications and design description	Validated OBSW and parameters settings
Equipment user manual	Validated system database
OBC model specification	OBSW performance and robustness analysis
OBC breadboard/prototype	
System database	

Table 6: SVF input/output

The main SVF's features are:

- SVF is constituted by a specific architecture configuration, equipment functional models, environmental and dynamics models, OBC models or EM, debugging tools, command consol;
- Different configurations are available: 1. Virtual model of the system and the environment and dynamics and OBSW algorithms (AIL), 2. Virtual model of the system and the environment and dynamics and OBSW compiled but not running in the flight boards (SIL), 3. Virtual model of the system and the environment and dynamics and OBSW running on the flight processor (CIL), and 4. OBSW running on flight processor, real HW equipment and virtual model of environment and dynamics (HIL);
- Setup: to specify, develop and integrate models in the different configurations, to integrate the OBSW algorithms with the other models, to integrate the EM of the OBC, to define scenarios and test procedures;

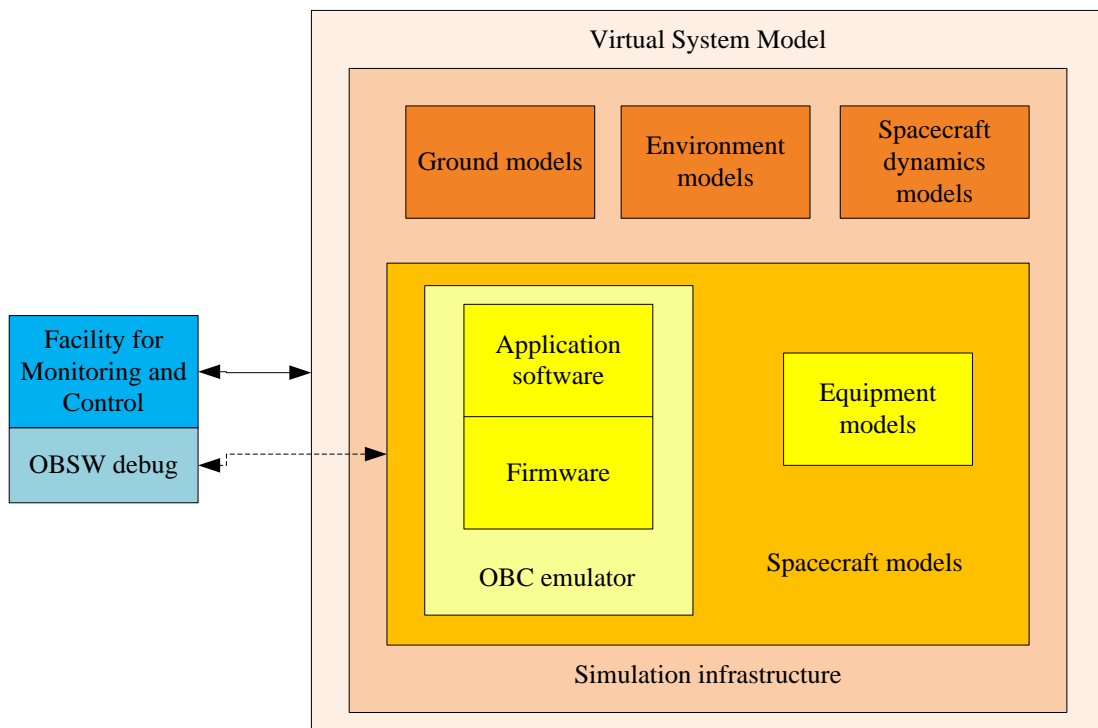


Figure 20: SVF - Software Configuration

- Simulator validation: the simulator should be validated without the availability of a validated OBSW. The various equipment models should be validated stand-alone in open-loop by

sending commands and verifying the proper update of related telemetry. This should be done using the appropriate simulated hardware interface: data bus, dedicated I/O etc. The validation is done wrt equipment data-package and according to the specified level of accuracy. The level of validation of the OBC software model should be increased by running cross-validation campaigns between the software models and a breadboard OBC. Also integrated tests should be done to validate the connections between a subset of models: e.g. sending a command to a power unit to power-on spacecraft equipment. Open loop tests should also be used to validate the orbit, environment and dynamics models and their integration with one another and the spacecraft equipment. If available, closed-loop test results from the FES at subsystem level (especially AOCS/GNC) can be used to validate the closed-loop behavior. Finally, the SVF validation campaign should cover the spacecraft models commanding/monitoring through TM/TC link, involving either a stub OBSW or an early version of it;

- Reuse: command consol and simulation core could be the same this and next phases. All the models of equipment, environment, and dynamics derive from the previous phases.

The SVF is used repeatedly during the program for each version of the onboard software and each version of the spacecraft database associated with it.

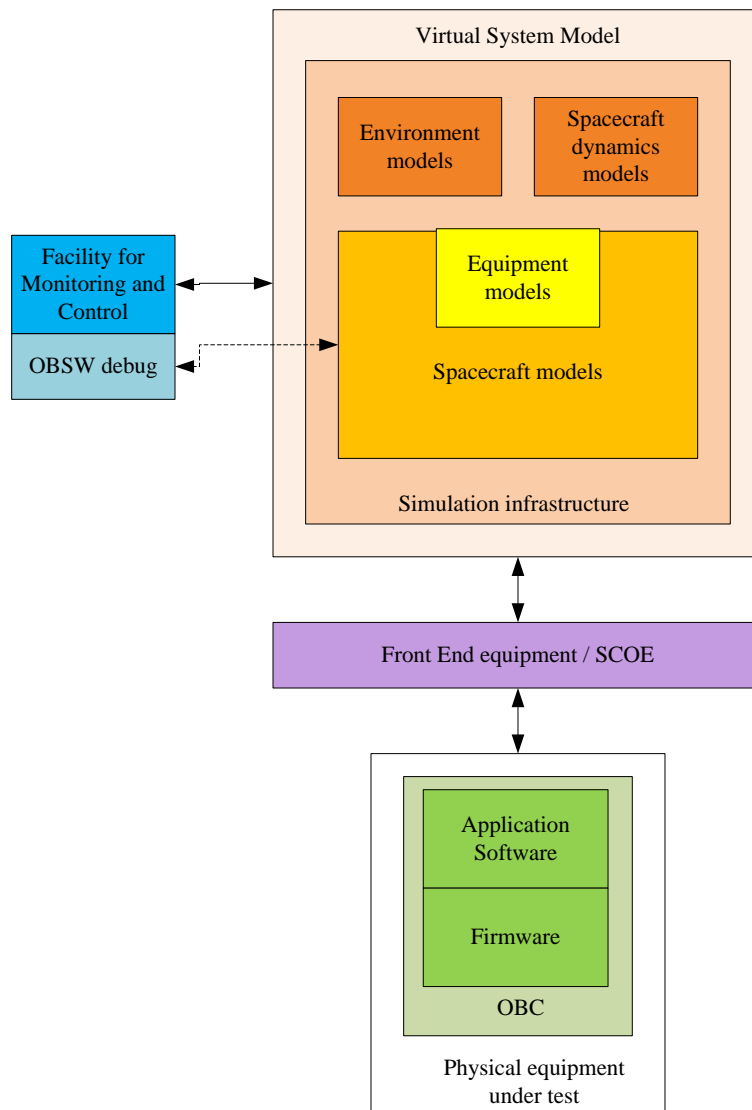


Figure 21: SVF- Hardware in the Loop configuration

1.4.2 Simulators for qualification and acceptance (AIV simulator)

In AIV phase, the simulator replaces the unavailable equipment and simulates the environment and the system dynamics. Focal issues are Real Time operations, closed loop tests in which the reaction to ground command and to the environment (through stimuli) are taken into account.

Spacecraft AIV facilities are the main simulators for qualification and acceptance purposes. The unavailable equipment is substituted by functions of the simulator, eventually located in another machine or sometimes in embedded systems. The used models have an increasing level of complexity, deriving from the “analysis and design” simulators.

The main objective of this kind of simulators is to support the incremental system V&V process. It should:

- be scalable in order to reduce the duration of the V&V campaign involving real hardware and GSE, an AIV Simulator can be setup to support the AIV campaign preparation.
- replace missing equipment in the course of the incremental spacecraft integration
- also simulate the system behaviour which cannot be represented on-ground (e.g. orbit and environment, appendage deployment) and will need to include as well the functional simulation as the simulation of the stimulation of real equipment (sensors) through dedicated GSE.

Input	Output
Validated OBSW	Integrated and validated equipment having undergone a validation test campaign should be available for the system integration
GSE user manual	The integrated and validated system having undergone the V&V campaign should be available to further V&V activities
Test procedures	

Table 7: AIV simulator input/output

The AIV simulator features are:

- The simulator is constituted by specific architecture, the GSEs and their interfaces, system models (whole or parts), models of dynamics, environment, and orbit.
- The simulator could be configured for AIL, SIL, CIL, HIL
- *Simulator setup activities*: it should need to specify the system AIV configuration plan in an effective way in order to verify the requirements according to the expected features and setting of the parameters, i.e the accuracy. The virtual model of the system shall be defined accordingly the kind of simulation session and the integration of the GSE shall be completed
- *Validation*: the simulator RT capabilities, GSE and the models derived from the previous simulator should be validated
- *Reuse*: The spacecraft equipment models included in this simulation facility should be those developed and incrementally validated in the frame of the SVF.

1.4.3 Simulators for qualification, test and operations of ground segment

In this case, the virtual models of the space system are involved in the validation of the ground segment and for users training. Operations Simulator has the main features of the previous types of simulators from which the used models, software, architectures, and configurations derive. The validation of the ground segment mainly passes for the verification of its capabilities and performances against the space segment in terms of SW and HW.

1.4.3.1 Ground Segment Test Simulator

The main objectives for a Ground Segment Test Simulator (GSTS) are:

- To verify each Ground System component in isolation against its requirements.
- To validate the Ground System to ensure that it supports the launch and operations activities including:
 - Support incremental Ground System integration;
 - Support end-to-end Ground System tests;
 - Support data-flow tests during operations.

This simulator allows tests, focusing on the telemetry and tele-command transfer and fault injection: in fact, the test foresee dataflow test (i.e. pre-pass data flow test which is a test performed routinely to check the TM and TC dataflow paths through the ground segment with the support of a spacecraft simulator prior to every spacecraft pass over a ground station, and Mission Readiness Tests (MRT), covering dataflow test at Ground Station checkout performed against the Mission Simulator).

They can be performed on single component in complete isolation from the others or on the complete system trying on configuration independent from the mission or dedicated to a specific mission. The space system simulation is clearly involved (in particular the SVF).

Input	Output
All models databases in standard format	Validation and verification of the Ground System
Ground System component documentation	

Table 8: GSTS input/output

The main GSTS features are:

- *Components*: System model focusing on the TM/TC data handling simulation as well as supporting the capability to interface the real system during the system validation test, Ground interface models both for simulate the RF interface and for provide the protocol, the specific chosen scenario and the configuration parameters of the simulator
- *Configuration*: the simulator could be configured in a software (the simulator software) configuration only or with various real elements in the loop
- *Setup*: it consists of develop any mission specific model behavior and apply the specific mission configuration
- *Validation*: the simulator is validated against the applicable standards in particular the TC/TM packet standards
- *Reuse*: the GSTS should as far as possible use standard simulation infrastructure. The models should be developed wrt the applicable standards to allow reuse between missions. Its models can be reused for the users training simulator

1.4.3.2 Training Operation and Maintenance Simulator

Training Operation and Maintenance Simulator (TOMS) shall help

- to ensure that the users/operators are ready to support the launch and operations activities (nominal and off-normal),
- to validate the operations procedures
- to support the trouble shooting and maintenance during operations

The simulator contains a high-fidelity model of the system and its ground segment interfaces, with an emphasis on providing a highly representative simulation of the spacecraft platform and payload

control housekeeping telemetry and telecommanding. The simulator should represent the behaviour for the spacecraft and its payload such that to the flight control team its *effects* in the telemetry are indistinguishable (as far as practicable) from the real spacecraft. The simulator should support the execution of the onboard software image(s) without modification. The simulator should also model the ground stations and network interface to allow direct connection to the mission control system. The simulator should support the injection of predefined failures by the operator in the space segment and ground segment. The simulator should be designed, developed and maintained to support the operations at least for the planned life-time of the mission.

Input	Output
Operation requirements including the reference to failure cases to be modelled, test/check point, validation criteria	Validated flight operations procedures
System specification and user manual	Trained users/operators in charge of the flight operations
Models databases of the system	Continued support during the real on-orbit operations through re-training, new procedures validations, anomalies investigation
GSEs and their manuals	
Flight operation procedures	

Table 9: TOMS input/output

The main TOMS features are listed hereafter.

- TOMS contains real time architecture configurations, system, subsystems and equipment model for Ground Segment and Space Segment, models of the environment, orbit and dynamics as well as the thermal and electrical behaviour, GSE and their interfaces, simulator scenario procedures.
- TOMS can be configured like SVF
- *Setup activities*: the following activities should be performed simulator system requirements definition, design, development and integration of the space segment and ground segment virtual/real model.
- *Validation*: TOMS should be validated against a representative set of GSE procedures and/ or flight operations procedures to verify that these procedures function as expected. This validation should be repeated with each phase delivery (each with successively increasing functionality).
- *Reuse*: the simulator should make maximum reuse of standard infrastructure and models (e.g. ground, environment, dynamics, thermal and electrical behaviour), models from previous missions with the same equipment and models from other simulators used in the mission (e.g. SVF, GSE). Moreover, TOMS can support ground segment V&V, onboard software maintenance (and can be reused as the basis for other simulators in subsequent missions).

1.4.4 In The Loop Configuration

The design and verification processes of any engineering system make use of modeling and simulation solutions. Several techniques have been developed in this field thanks to the dramatic progress of technology in the last decades. A complete simulation facility shall be designed in order to perform

with different architecture. A general categorization of the simulation configuration is proposed in the following sub-paragraphs, according to Eickhoff [28].

A few important stages of interaction between the elements of the system along this process can be defined:

- 1) Algorithm in the loop (AIL). The algorithms of interest are added to the pure numerical simulation. They are not yet written in the formal language that will be used on the final hardware and are not run on it. AIL is mainly used at design stages with the objective of testing the algorithms
- 2) Software in the loop (SIL). Algorithms are translated into the final programming language, but they run on ground hardware. The software carries out all the required functions, but in general its performance is different with respect to running it on a flight unit architecture (e.g. PC vs embedded-PC)
- 3) Hardware in the loop (HIL). Real hardware is included in the simulation loop, and consists typically of sensors and/or actuators. HIL is a hybrid software-hardware simulation architecture, in which the hardware part can vary from a few pieces to the fully integrated system. HIL technique is particularly useful for the verification of all those elements that operate in special environments and conditions which are difficult to reproduce in a laboratory. It may help to detect unexpected behaviors and/or failures arising from the integration of the component in the global system.

1.4.4.1 Algorithms in the loop

In an initial step, the physics of the system is modeled and the developed control algorithms are integrated to control the system. The algorithms mostly are not yet implemented in the target programming language, neither on targeted hardware. In other words, a first complete functional model related to the system or component or parts of it and of the space environment is available.

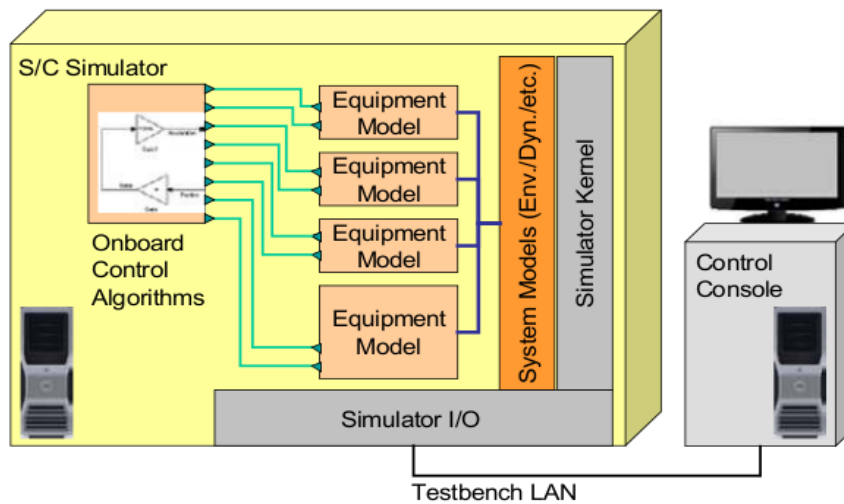


Figure 22: Algorithms In The Loop configuration [28]

1.4.4.2 Software in the loop

At this time, the algorithms are coded in software in the target language. The now available control software is loaded onto the - eventually modified – test stand, again, to control the system. A detailed on-board computer simulation should be available allowing on-board software.

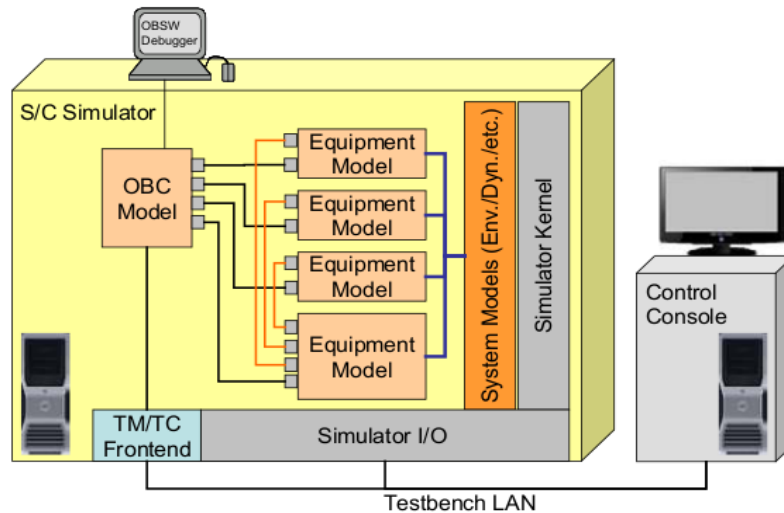


Figure 23: Software in the Loop [28]

1.4.4.3 Controller in the Loop

The microprocessor/controller is at this point available in hardware. The control software is loaded onto a representative target computer, which now controls the hybrid test stand. The final software on the target computer now has simulated system physics. Tests of compatibility between software with hardware and tests of on board software with simulated satellite are carried out.

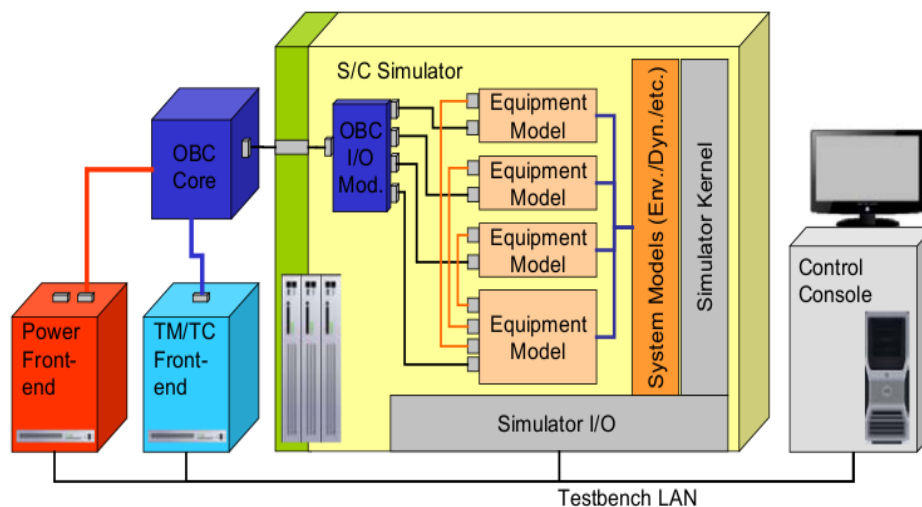


Figure 24: Controller in the Loop [28]

1.4.4.4 Hardware in the loop

One effective modeling and simulation method is the Hardware-In-the-Loop (HIL) approach. This methodology consists in the combination of computer simulation and hardware in a single coherent platform [29]. Typically, HIL simulation is the last stage of the testing and integration process of an engineering system.

In the literature there are some examples regarding simulation based on HIL technology. A wide range of options exists in terms of level of complexity. An example of a complex HIL architecture is the simulation of on-orbit docking between two spacecraft at the Spacecraft Robotic Laboratory of the Naval Post Graduate School in Monterey, CA [30]. In this facility, the target and the chaser are physically reproduced, while the rest of the system is simulated on a real-time simulation computer. In [31], the authors show the verification campaign carried out for the Special Purpose Dextrous

Manipulator installed onboard the International Space Station. In this case, the space hardware is simulated and the contact dynamics is emulated using a rigid robot. In [32], the authors describe a ground-based HIL simulation facility for Rendez-Vous and Docking (RVD), called European Proximity Operations Simulator (EPOS), at German Space Center. The laboratory is aimed at providing test and verification capabilities for complete RVD processes of on-orbit servicing missions using two robots for the physical real-time simulation of the maneuvers. Other examples of HIL simulation can be found in literature applied to non space-related systems, as described in [33] – [29]. HIL technique can be applied for verification from component to element level. In [40], HIL simulation is used for the development of a network of sensors for pico-satellite missions. This work underlines how special features of the network are investigable only via the HIL approach. Another interesting example is the development of a HIL simulator for the simultaneous test of an Attitude Heading Reference System and an Attitude Control System based on momentum wheels [41]. Finnset et al. dealt with the simulation of the attitude control system of a small satellite, the European Student Moon Orbiter, using hybrid SIL-HIL architecture [42]. All these applications show the effectiveness of a test campaign conducted via HIL simulation.

Hardware In The Loop simulation aims to make the software on the target hardware now control the real system, and no longer the test stand's system simulation. The simulators here are required for computations of stimuli parameters to reflect gravity-free space conditions to reflect the microgravity space conditions, as well as magnetic field changes and so on.

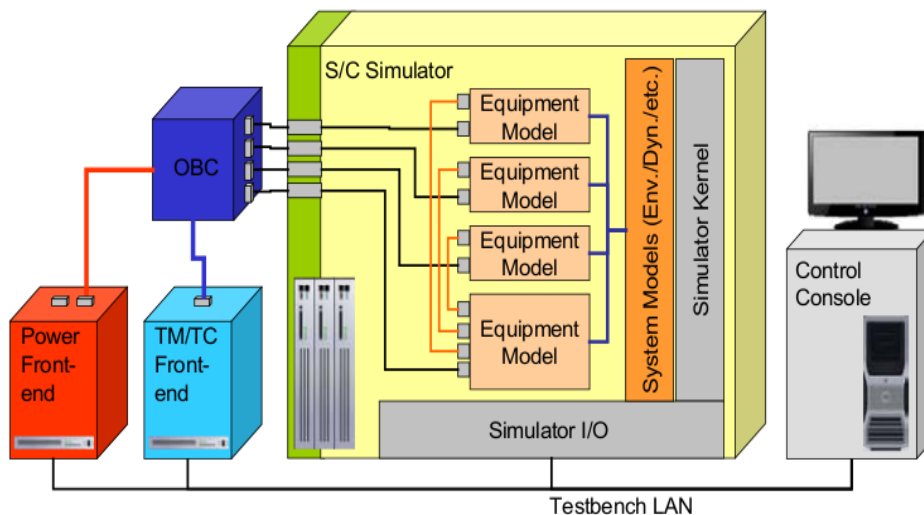


Figure 25: Hardware in the Loop [28]

All these kinds of simulation based system development approach requires fundamentally new workflow processes to be applied, both concerning applied technology as well as with respect to project organization and distribution of responsibilities. Among the main aspects that have to be managed, it is possible to underline:

- the integration of engineering disciplines such as mechanics, electrics, system operations / data handling,
- the definition, the requirement compliant development and qualification and installation of the simulator.
- the consistent application of system models, configuration databases and test procedures overall project phases
- the standardization of the simulator environment in order to reuse qualified “elements” in future projects

- the process consolidation for the integrated development and test of GNC hardware and software, and the simulator

1.5 Models and the modelling process

A general and univocal definition of model is quite difficult to provide because different aspects concur to the model design and construction.

A generic model represents an entity that can be configured to represent any instantiation of that entity.

The process of system modeling typically involves an analysis phase followed by a subsequent design phase. In the analysis phase, the key aspects are represented by questions such as “What is the problem space? What are the envisioned components and how are they related to each other? What are the attributes and the operations of the components and how are they interacting in order to accomplish the intended result? etc.” In this context, the design of a system usually involves a structural representation of the components and their relations, along with a behavioral specification that captures the system dynamics.

Consequently, when modeling a system, the design can be viewed generally from two main design perspectives, namely the structural description and the exhibited behavior. By simulation models it is meant both data models, e.g. geometrical model of a system, and behavioral models, e.g. the algorithms representing the behavior of a component or environment expressed in a high level programming language.

The **structural perspective** can be captured specifying the distinctive attributes of the system and its components. Alongside, the user can specify their respective relations with respect to each others. Structural analysis helps to evaluate numerous quality attributes and may be used as a feedback in many tuning and optimization mechanisms.

The **behavioral perspective** can be encoded in order to capture the dynamics of various state parameters in the system or the underlying components. Furthermore, in the behavior perspective, the different internal or external interactions of the system shall be taken into account. The behavioral analysis is usually much more demanding and involves intense rigor and precision.

1.5.1 Model architecture

Generally, but not always, each model has inputs and outputs and internal states. A complete representation of a model is proposed in Figure 26.

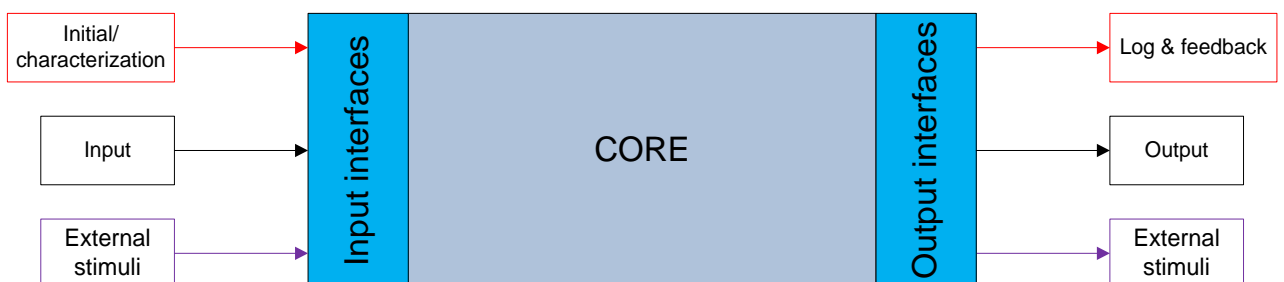


Figure 26: Model architecture

Some considerations can be made in order to describe the figure:

- The core of a model depends on the type of the considering model (virtual vs hybrid vs physical). In fact, it can contain the physical modeling or the mathematical equations that explain the real behavior (e.g. the basic Newton’s law, $F=ma$), the solvers (e.g. integrators or

derivative methods), the work logic (i.e. change of mode during the operation, for example the activation and deactivation of elements or functions according to external situations).

- Model outputs are all the parameters computed in the model core execution and describe the behavior of simulated equipment or phenomenon as well as the actual response of a physical model subject to specified input. They can be provided to the simulator or the test object.
- Both inputs and outputs shall be formatted according data exchange protocols and rules, particularly for the SIL and HIL.
- Characterization inputs channels serve to calibrate parameters and define constant values characterizing the models behavior. E.g. the calibration settings of a sensors, or general mission/system parameters that do not change during the simulation, for example the gravitational constant for a satellite in LEO.
- External stimuli can be outputs of virtual model that will be used to set the GSE connected with the object under verification as well as inputs of physical models that receive the stimulation reproducing the real conditions. Moreover, these channels can be used to inject external failures into the model.
- Log/feedback outputs are channels dedicated to the log/store the characteristic parameters related to the model, from which analyze the simulation evolution live or after the end of the session

1.5.2 Level of detail and degree of fidelity of a model

“A simulator’s scope refers to its breadth”[28]: it means that the capabilities of the simulator is strictly related with what subset of the real world it is able to represent, with the number and the quality of its models for every real-world component or phenomenon, with the number and type of interactions and interfaces of the model that carry out to easily aggregate components and treat them as large elements. Clearly, the most detailed, complex simulators are broad and highly resolved. However, a deep difference passes between detail and fidelity. It is natural to think that modeling everything in excruciating detail will most closely match real performance: often it is wrong and detrimental. Sometimes a detailed simulator just turns out wrong, whereas a simpler, more easily understood simulator closely matches real performance.

Details may even produce “noise” that keeps away the user from seeing simple relationships. Adding detail can also demand more data, and the data may be wrong. Cause-effect relationships among objects in a simulator are crucial. The trick is to identify details that matter and ignore ones that don’t affect the results: following this way the simulator’s outputs assess an increasing degree of fidelity. The tendency is to simulate as much detail as possible given the time available, even if some of that detail is unnecessary.

The choice of a model and its level of detail changes in relation to:

- **the product life cycle phase.** in the first phase of a SE process the virtual models are the only models involved in the simulation activities. Their core consists of few basic rules and equations that explain the main behavior of the modeled element, with low details and a degree of fidelity sufficient to make feasibility studies and have an idea of the evolution and the quantity of the main parameters for a preliminary sizing. During the design phases, the global architecture of the system and the environment is better rather than the details of a single model (e.g. the sensors, the actuators and the electrical devices as well as the effects of environment phenomena models update the global model wrt the feasibility phase). The control is based on model simplifying both the controller and the observer design (e.g. the design of the controller is made thanks to linearized equation of the system and verified on the non linearized model). In the verification phase, the level of detail of the models increases

because all the equipment are well defined, the real capabilities of the system are investigated against the actual environmental conditions, and the control performances shall be guaranteed for different situations (e.g. disturbances action). The last system model has the highest possible degree of fidelity.

- **the supportable computation complexity.** This issue depends on the features and performances of the simulation machine. As said before, a higher level of detail requires higher resources for support the operations of complex models, based on high order equations, accurate solvers and meticulous definition of the work logics. All these characteristics tend to decrease the speed of simulation and they can compromise real time activities. That leads to pay attention about the definition (for the developers) and the choice (for the users) of the detailed models, especially when SIL, CIL, and HIL are performed.
- **the required accuracy of the results.** Complex and detailed models generally produce accurate results: it means that the values obtained for the state variables and other main parameters of the system take evolutions and trends that reflect the real behavior. However, apart for the validation purposes, the best accuracy is not necessary. Moreover, in relation to running session, particular performances and functions are more investigated than others: it should allow to the user the selection of models with various levels of detail within the same simulation architecture in order to reach different degrees of fidelity within the same run. For example, evaluating the performance of a chaser in the mating phase, position and attitude accuracy become the focus features that should be available with the highest fidelity for the users and analysts. At the same time it should be negligible to investigate the fuel consumption (and consequently the sloshing whose model can be at lowest level of detail or, even, removed).
- **the costs & resources:** virtual models cost and require less human resources and items with respect to the physical ones. Programmatic requirements address and constraint the choice of the models
- **the instrumentation and facilities availability.** The choice depends on the availability and the in-house models. In particular for physical models (also for the virtual) it can be very cost effective and quick to use already available models wrt develop it as new. This is also because the model is just tested, integrate and, probably, validated with the simulator.
- **the parametrization.** One of the main goals for the model designer is to produce a flexible model and the parameterization are the main features that shall be sought. Parameterization is the process of deciding and defining the parameters necessary for a complete or relevant specification of a model. Most often, parameterization is a mathematical process involving the identification of a complete set of effective coordinates or degrees of freedom of the model, without regard to their utility in some design. In other words, the parameterization carries out to define constants and variables characterizing a model: a priori settable parameters in the case of constants (as calibration values) or parameters that evolve in time and determine the behavior of the model. Changing constant parameters, the intrinsic features of a model vary; changing variables parameters change the entire model property.

1.5.3 Type of models

Depending on the context, the models can be classified according to their fidelity, their domain or their modeling technique.

1.5.3.1 *Virtual model*

A virtual model is a set of instructions, rules, equations, or constraints that virtually reproduce a phenomenon or a system behavior. In this case, the model “is a copy of an object” [44].

Virtual models (or mathematical models, simulators) could be exploited to simulate a system dynamics, to train ground support operators, mathematical model to optimize system definition and layout, to analyze structure dynamics and its strength at launch. Moreover, virtual models could be used for thermal analysis, mechanical deployment simulation, electromagnetic compatibility, harness routing configuration and attitude control system evaluation. These models reveal to be more and more useful to save time and money. Indeed, they can be adapted to further mission and space systems analysis and verification.

A virtual model approach can be applied in every project phase, first of all in Phase A and results an incentive to the data exchange and the sharing between the projects actors: using standard allow a more quickly data exchange and generation of new models.

A virtual representation of the system (and the mission), allows

- the validation and optimization of the project
- to start the implementation with a sufficient level of detail.

Consequently, the keys of the project can be individuated and addressed in the right way, avoiding an expensive redefinition in the next phases.

A shared virtual model allows to work together and contemporary to different teams and to cooperate also in remote, minimizing efforts and resource usage/wasting.

In a traditional verification process, the test activities are strictly related to the physical construction of the system: each component and sub-system is individually tested, then integrated and tested together other system elements. Using a virtual models and M&S based approach this step is previously simulated so that the procedures can be generated and pre-validated. At the same way, the integration can be virtually verified in parallel to the development of the single component.

1.5.3.2 *Physical models*

1.5.3.2.1 Mock-ups (MU)

Mock-ups (MU) are representative from mechanical, geometrical and physical properties point of view. It is useful in support of design definition for overall architecture analyses, configuration design and assessment, interface control and definition, human factors and human computer interface (HCI) assessment, operational procedures evaluation and layout optimization. It is progressively upgraded to redirect a final configuration according to the design evolution.

1.5.3.2.2 Engineering Model (EM)

Engineering Model (EM) represents in form, fit and function for the flight unit, generally without redundancy and space qualified parts. It is representative for physical and electrical configuration (including interfaces). The EM is used for functional validation and failure survival demonstration. EM is involved in functional tests at standard ambient condition, helps to validate final software, test facilities, GSE and, eventually, assembly, integration and test procedures.

1.5.3.2.3 Qualification Model (QM)

Qualification Model (QM) fully reflects the end product design in all aspects, including manufacturing methods and procedures. It is used for complete functional and environmental qualification tests. This model will be subjected to testing up to the limits of the operational and non-operational environmental conditions specified for the instrument to endure. The parts used will be of the same type and specification as the models built to fly on the spacecraft, and will be parts specially designed to survive in the harsh space environment.

1.5.3.2.4 Flight Model (FM)

Flight Model (FM) is the flight and product. It is subjected to formal functional and environmental acceptance testing. It is validated mainly by similitude with QM, in fact FM is built according to the QM configuration. Acceptance levels are wider than expected operating conditions, but not as extreme as Qualifica

1.5.3.2.5 Prototype vs proto-flight philosophy

A prototype is fully representative of the FM, even though its components are not qualified for space application. This approach, which consists on implementing separately EM, QM and FM, is generally used in projects for which all affordable measures are taken to achieve minimum risk. The usual characteristics of these projects are:

- new or complex design,
- impossibility to be recovered or repaired after launch,
- special mission requirements.

The prototype approach makes extensive use of the aforementioned defined models to cover verification needs (environmental qualification, functional and performances validation, GSE validation, interfaces and AIT procedures verification). The disadvantage is high cost and project schedule, which has to allow FM integration only after having qualified QM.

The advantages of this approach are:

- low risks,
- capability to perform parallel activities on different models,
- completion of qualification activities prior to acceptance,
- capability to use QM or EM as integration spare during higher level activities,
- FM is not subjected to qualification tests but only to flight acceptance tests.

The risks of this approach are:

- EM and QM representativeness,
- If QM is fabricated very early, then qualification test are completed on FM,
- Delay in delivering QM may affect the whole mission time schedule.

On the basis of project requirements, the related model philosophy is tailored. A common case is to use, after the thermal and structural qualification, parts of the STM to complete the space item level EM, in addition to EM/QM equipment. It is important to note that after the system electrical or functional qualification the EM is used for ground support to flight operation. Feedback from qualification on the FM manufacturing is also taken into account. In addition, after system functional, thermal and structural interface verification between elements, QM and STM are used for flight simulation on ground.

A protoflight is fully representative of the FM and its components are qualified for space application. It goes through qualification and flight acceptance tests.

This approach is applied to projects whose characteristics are:

- no critical technology is employed in the design,
- qualified products are extensively used, and
- compromises are permitted to reduce cost by accepting a moderate level of risk.

The pure protoflight approach is based on a single model to be flown after it has been subjected to a protoflight qualification and acceptance test campaign (see [8] for details).

The advantage of this approach is its lower cost. The disadvantages are:

- increased risk since the FM is subjected to qualification level test (fatigue and stress are accumulated on ground),
- serial activity flow on the same model,
- mixed qualification and acceptance activities, and
- no integration spares.

The risks of this approach are:

- late qualification, which may cause development risks. This requires to implement a prototype philosophy on the most critical components,
- too large margins level are taken into account (over-sizing issue),

tion levels.

1.5.3.3 *Hybrid models*

Hybrid models are a combination of both virtual and physical models. They are crucial elements in the V&V process because permit to develop and verify step by step software and hardware along the project.

The hybrid model philosophy is used in projects where advanced qualification activities are performed in areas of new design or in areas having a critical impact on the verification program. The hybrid approach always results in a protoflight model be flown after a protoflight test campaign whose scope is reduced with respect to that of the pure protoflight approach. Specific qualification tests in the critical areas are carried out on dedicated models. In these areas only acceptance testing is performed on the Proto-Flight Model (PFM).

The advantages and disadvantages of this approach lie between those of the prototype and the protoflight approaches in terms of risks, costs and schedule. It represents a good compromise; in fact this is the reason why it is often selected. In particular, in the hybrid approach it is feasible to:

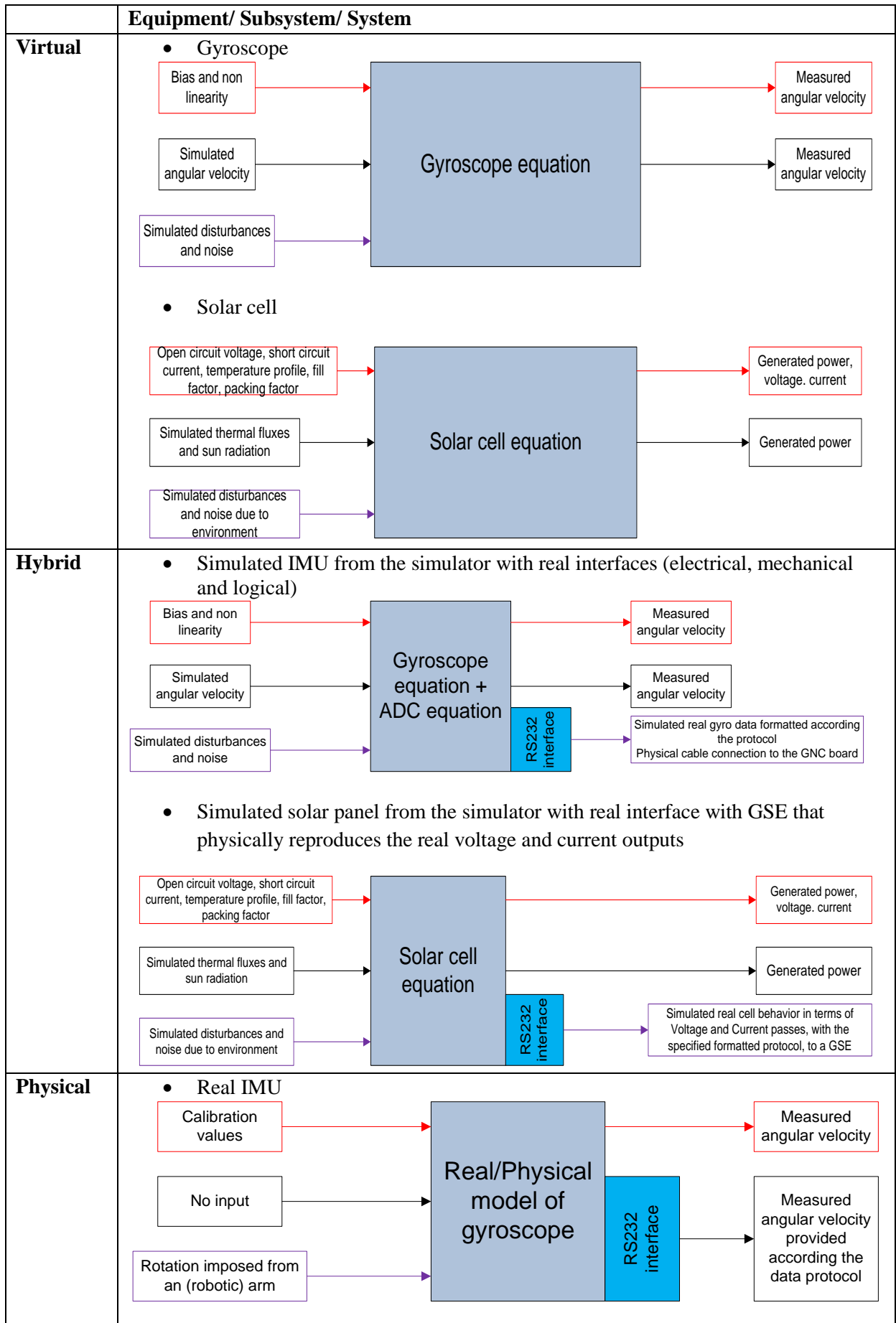
- perform some parallel activities
- use QM as integration spares during high level activities,
- conform to the delivery dates of high reliability components and accommodate possible use of commercial components.

It is important to note that:

- the decoupling of the STM (SStructural and Mechanical) activities from the EM activities enables program flexibility and reduction of schedule risks,
- the EQM or PFM is qualified at equipment level, depending on its development status,
- a suitcase model and the software validation facility at satellite level can be used to verify specific interface performance,
- a mock-up structure can be used for the EM configuration.

The models philosophy defines the optimum number and the characteristics of virtual and physical models required to achieve confidence in the product verification with the shortest planning and a suitable weighing of cost and risks. It consists of developing models in a preliminary or detailed way in order to perform verifications.

The categorization proposed in this thesis can be summarized in Table 10 and Table 11 through some example.



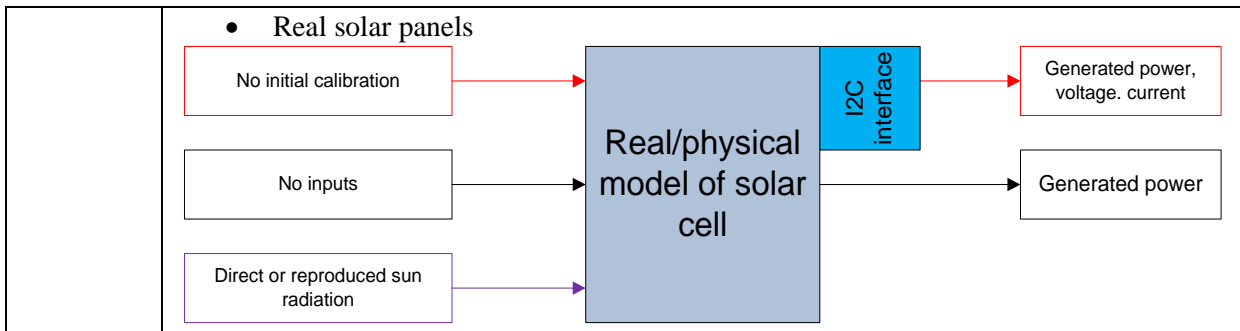


Table 10: Examples of equipment/subsystem models

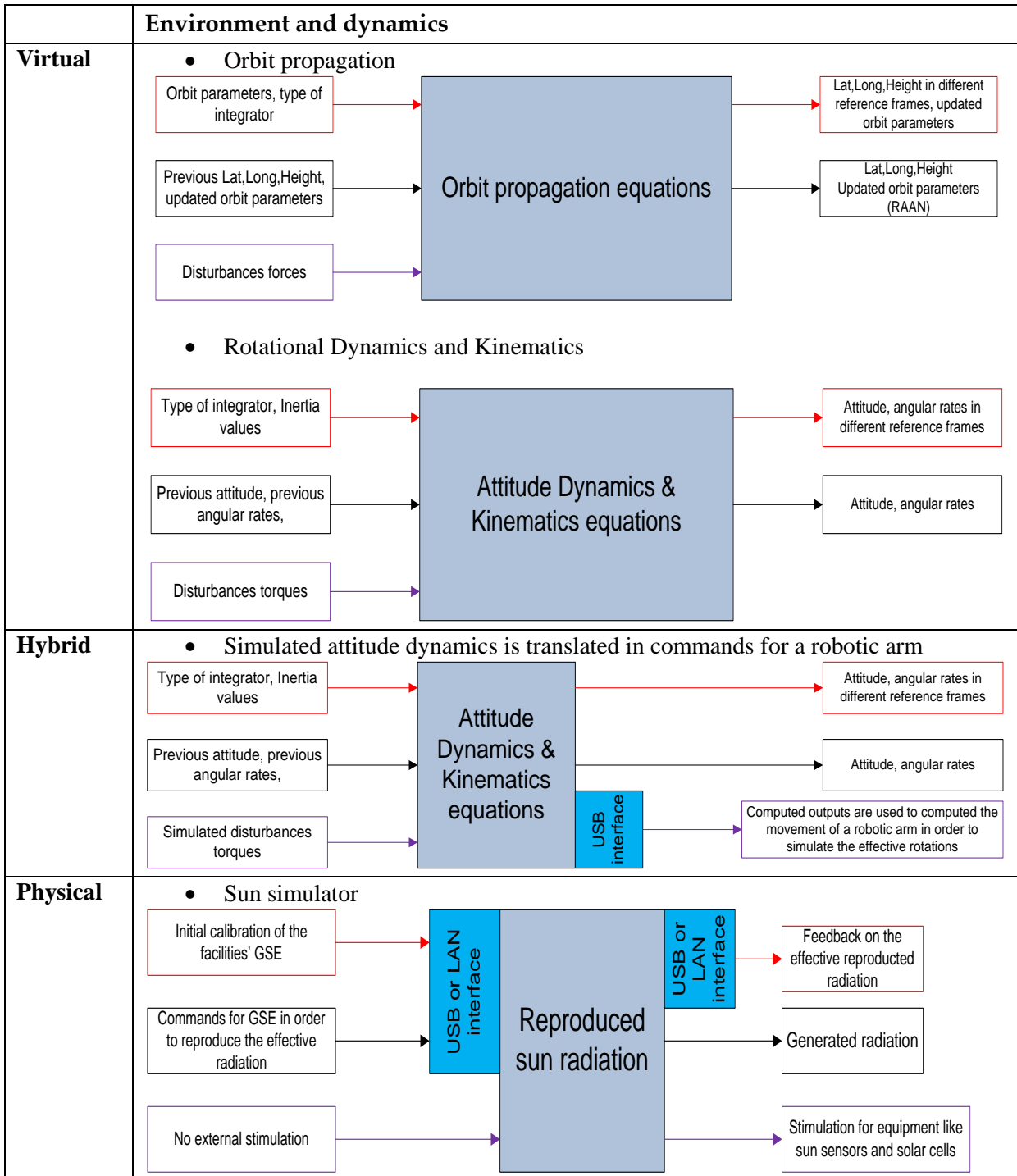


Table 11: Examples of dynamics and environmental models

Chapter 1 reference

1. ECSS-M-ST-10C - *System Engineering –General Requirements* (06/03/2009)
2. *Systems Engineering Handbook NASA/SP-2007-6105 Rev1*, National Aeronautics and Space Administration, NASA Headquarters,2007
3. Fhese, *Automated Rendezvous and Docking of Spacecraft*, Cambridge University Press, 2003, ISBN 0 521 82492 3
4. D.A Cook and J.M. Skinner. *How to Perform Credible Verification, Validation, and Accreditation for Modeling and Simulation*. In Special Systems & Software Technology conference Issue, CrossTalk, The Journal of Defense Software Engineering, vol. 18(5) May 2005.
5. ECSS-E-ST-10C – *Space Engineering: System Engineering General Requirements-01/06/2009*
6. ECSS-E-ST-10-02C – *Space Engineerig – Verification* (02/09/2009)
7. L. B. Rainey, *Space Modeling and Simulation Roles and Applications Throughout the System Life Cycle*, The Aerospace Press El Segundo, California American Institute of Aeronautics and Astronautics, Inc. Reston, Virginia, 2004
8. ECSS-E-ST-10-03C- *Space Engineerig – Testing* (01/06/2012)
9. ECSS-E-TM-10-21A – *Space Engineering –Space modelling and simulation –* (16/04/2010)
10. Mourad Debbabi, Fawzi Hassaine, Yosr Jarraya, Andrei Soeanu, Luay Alawneh *Verification and Validation in Systems Engineering (Assessing UML/SysML Design Models)*, Springer, 2010
11. *Object Management Group. About the Object Management Group (OMG)*. <http://www.omg.org/gettingstarted/gettingstartedindex.htm>. Visited: December 2006.
12. *International Council on Systems Engineering (INCOSE) Website*. <http://www.incose.org/>.Last Visited: February 2010.
13. Object Management Group. *MDA Guide Version 1.0.1*, June 2003
14. INCOSE. *Systems Engineering Vision 2020*. Technical Report TP-2004-004-02, International Council on Systems Engineering (INCOSE), September 2007.
15. S. Ceri, P. Fraternali, and A. Bongio. *Web Modeling Language (WebML): A Modeling Language for Designing Web Sites*. Computer Networks (Amsterdam, Netherlands: 1999), 33(1–6):137–157, 2000.
16. D. M. Nicol, *Special Issue on the Telecommunications Description Language. SIGMETRICS Performance Evaluation Review*, 25(4):3, 1998.
17. D. Agnew, L. J. M. Claesen, and R. Camposano, editors. *Computer Hardware Description Languages and their Applications*, volume A-32 of IFIP Transactions. North-Holland, Amsterdam, 1993.
18. Object Management Group. *Introduction to OMG’s Unified Modeling Language(UML)*. http://www.omg.org/gettingstarted/what_is_uml.htm. Last visited: December 2006
19. T. Weilkiens. *Systems Engineering with SysML/UML: Modeling, Analysis, Design*. Morgan Kaufmann Publishers Inc., Burlington, MA, 2008.
20. H.P. Hoffmann. *UML 2.0-Based Systems Engineering Using a Model-Driven Development Approach*. CROSSTALK The Journal of Defense Software Engineering, November 2005.
21. L. Doyle and M. Pennotti. *Systems Engineering Experience with UML on a Complex System*. In CSER 2005: Conference on Systems Engineering Research, Department of Systems Engineering and Engineering Management, Stevens Institute of Technology, 2005.
22. TASTE - http://taste.tuxfamily.org/wiki/index.php?title=Technical_FAQ

23. F. Pasquier, G.Cantenot, *SIMSTB: A full Software SVF in the Frame of the MDV Approach*, Proceedings of the DASIA 2005 conference: data systems in aerospace Conference, Proceedings of the DASIA 2005 conference: data systems in aerospace. ISBN [9290929138](#)
24. www.stk.com
25. J. Eickhoff; R.Hendricks; J.Flemmig: *Model-based Development and Verification Environment*, 54th International Astronautical Congress, Bremen, September 29th - October 1st, 2003
26. H. Eisenmann, C.Cazenave, *SimTG: Successful Harmonization of Simulation Infrastructures*, 10th International Workshop on Simulation for European Space Programmes,
27. <http://www.esoc.esa.de/external/mso/simsat.html>
28. J. Eickhoff, *Simulating Spacecraft Systems*, New York: Springer, 2009.
29. J. A. Ledin, "Hardware in The Loop Simulation", *Embedded Systems Programming*, Vol.12(2), pp. 42-60, 1999. https://edit.ethz.ch/idsc/Courses/embedded_control_systems/Exercises/Hardware-in-the-Loop.pdf, last access July 2013.
30. J. S. Hall, M. Romano, "Laboratory Experimentation of Guidance and Control of Spacecraft During On-orbit Proximity Maneuvers," in *Mechatronic Systems Simulation Modeling and Control*, 2010: pp. 187-225
31. Milella, D. Di Paola, G. Cicirelli, A. Distanti, *An autonomous mobile robotic system for surveillance of indoor environments*, International Journal of Advanced Robotic Systems. vol.7 (1), pp. 19-26, 2010
32. J. de Carufel, E. Martin, J.C. Piedboeuf, "Control strategies for hardware-in-the-loop simulation of flexible space robots", *IEE Proc. Control Theory and Applications*, 2000, vol.147(6), pp. 569-579.
33. T. Boge, T. Wimmer, O. Ma, M. Zebenay, "EPOS: A Robotics-Based Hardware-in-the-Loop Simulator for Simulating Satellite RvD Operations", i-SAIRAS, Sapporo, Japan, 2010.
34. R.Isermann, J. Schaffnit, S.Sinsel, "Hardware-in-the-Loop simulation for the design and testing of engine-control systems", *Control Engineering Practice*, vol. 7(5), pp. 643-653, 1999.
35. E.R. Mueller, "Hardware-in-the-loop Simulation Design for Evaluation of Unmanned Aerial Vehicle Control Systems", *AIAA Modeling and Simulation Technologies Conference*, Collection of Technical papers, 2007, vol. 1, pp. 530-543, Hilton Head, SC.
36. N.R. Gans, W.E. Dixon, R. Lind, A. Kurdila, "A hardware in the loop simulation platform for vision-based control of unmanned air vehicles", *Mechatronics*, vol. 19, pp. 1043-1056, 2009. doi: 10.1016/j.mechatronics.2009.06.014
37. E. Tara, S. Filizadeh, E. Dirks, "Battery-in-the-Loop Simulation of a Planetary-Gear-Based Hybrid Electric Vehicle", *IEEE Trans. Veh. Technol.*, vol. 62(2), Feb, 2013. doi 10.1109/TVT.2012.222.6921
38. D. Bullock, B. Johnson, R. Wells, M. Kyted, Z. Li, "Hardware In the Loop simulation", *Transportation Research Part C: Emerging Technologies*, vol.12(1), pp. 73-89, 2004.
39. X. Wu, T. Vladimirova, "Hardware-in-Loop Simulation of a Satellite Sensor Network for Distributed Space Applications", *Conference on Adaptive Hardware and System*, NASA/ESA, 2008, pp. 424 – 431.
40. D. Kim, S.Y. Park, J.W. Kim, K.H. Choi, "Development of a Hardware In-Loop (HIL) Simulator for Spacecraft Attitude Control Using Momentum Wheels", *Journal of Astronomy and Space Sciences*, vol. 25(4), pp. 347-360, 2008.
41. R. Finnset, S. K. Rao, J. Antonsen, "Real time hardware-in-loop simulation of ESMO satellite attitude control system", *Model Identification and Control Journal*, vol. 27(2), pp. 125-140, 2006.

42. ECSS-E-ST-10-03C *Space Engineering: Testing* – 01/06/2012
43. T. Wimmer, T. Boge, Q. Muehlbauer, *EPOS: A Hardware-in-the-Loop Robotic Simulation Assembly for Testing Automated Rendezvous and Docking GNC Sensor Payloads*, 8th International ESA Conference on GNCS, Karlovy Vary, CZ, 2011
44. H. Zipfel, *Modeling and Simulation of Aerospace Vehicle Dynamics*, American Institute of Aeronautics and Astronautics, 2000, ISBN: 1563478757
45. M. Pollina, Y. Leclerc, E. Conquet, M. Perrotin, G. Bois, L. Moss, *The ASSERT Set of Tools for Engineering (TASTE): Demonstrator, HW/SW codesign, and future*.

Chapter 2. GNC system design

2.1 What are G(Guidance), N (Navigation) and C (Control)?

The Guidance Navigation and Control system is one of the leading and most complex spacecrafts' subsystems. Its name is closely linked to the three main functions performed by the subsystem, but often in the literature is split into two other subsystems: ADCS (Attitude Determination and Control Subsystem) and ODCS (Orbit Determination and Control Subsystem).

In [1] *Navigation* or orbit determination (interchangeably) means determining the spacecraft's position and velocity or, similarly, its orbital elements as function of time. *Guidance* and orbit control (for satellite) means adjusting the orbit to meet some pre-determined conditions. Finally, *Control* refers to a shortened form of attitude control. ADCS stabilizes the vehicle and orients it in a desired direction during the mission despite the external disturbance torques acting on it, ODCS performs navigation and guidance functions, determining and control the orbit and, more in general, the trajectories.

In [2], authors specifies that the "motion of a rigid spacecraft is explained by its position, velocity, attitude and attitude motion. The first two quantities describe the translational motion of the center of mass of the spacecraft and are subjected to what is variously called as celestial mechanics, orbit navigation, or space navigation, depending on the aspect of the problem that is emphasized. The last two quantities describe the rotational motion of the body of the spacecraft around the center of mass. They point out "in general, orbit and attitude are interdependent. [...] However, we will normally ignore this dynamical coupling and assume that the time history of the spacecraft position is known and has been supplied by some process external to the attitude determination and control system".

In [3], *Navigation* is the determination of the current state of motion of the vehicle that is position, velocity and attitude. Navigation computes the current state using data of sensors that detect physical quantities: state-defining variables are often not directly measured. *Guidance* is the computation of corrective actions necessary to change from the navigation-determined vehicle state to a required vehicle state. In detail, guidance is the process of comparing the measured navigation state with the required one and then computing commands to correct the difference between them. The state vector is conveniently broken into two distinct parts that take account of the two natures of motion: translational motion for position and velocity, and rotational motion for attitude and angular velocity. *Control* is the application of corrective manoeuvres to obtain the change computed by guidance. The corrective manoeuvres refer to forces (and/or torques) that have to be applied to the spacecraft in order to drive the state variables to required values.

For the purposes of this thesis, the GNC system is considered as consisting of ADCS and ODCS. This choice starts from the idea that for a satellite ADCS and ODCS projects are decoupled, in particular small satellite often do not have to perform orbit/trajectory determination and control. In any case, sensors and algorithms for the determination, as well as the actuators used by the two systems are different. In the same way, for RVD/B missions the determination and control of orbit/trajectory and attitude follows different approaches both in absolute reference guidance and in relative guidance. Surely the interactions between the two subsystems shall be taken into account for the respective projects but they can be carried out independently.

In addition, the problems of orbit control/determination and attitude control/determination have different backgrounds. Predicting the orbital motion of a celestial body is a quite old science (i.e. Newton laws). In contrast, most of the advances in attitude determination and control have occurred since the launch of the Sputnik (1957).

2.2 GNC interactions with other subsystems

The design of the GNC is highly conditioned by the mission, the system and other subsystems design that impose constraints. Requirements derive from the mission analysis: the pointing accuracy of a payload, the time and the size of the manoeuvres to reach a desired attitude or orbit (also relative), the target to RV, the final orbit for deep space explorations, and the landing target for planet exploration. The mission affects the autonomy for the determination and the control of attitude and trajectory. Mission duration influences the choice of the actuators and sensors, and, consequently, the available electrical power and fuel.

Moreover, GNC shall satisfy special requests from other subsystems:

- Thermal Control System (TCS) may require to perform manoeuvres to prevent over/under-heating;
- Electrical Power System (EPS) may require the pointing of solar panels;
- Communication System (ComSys) may require the pointing of antennas and/or maintaining an orbit (station-keeping);
- Payload may require the pointing of the instruments and/or maintaining an orbit (station-keeping), e.g. meteorological satellites.

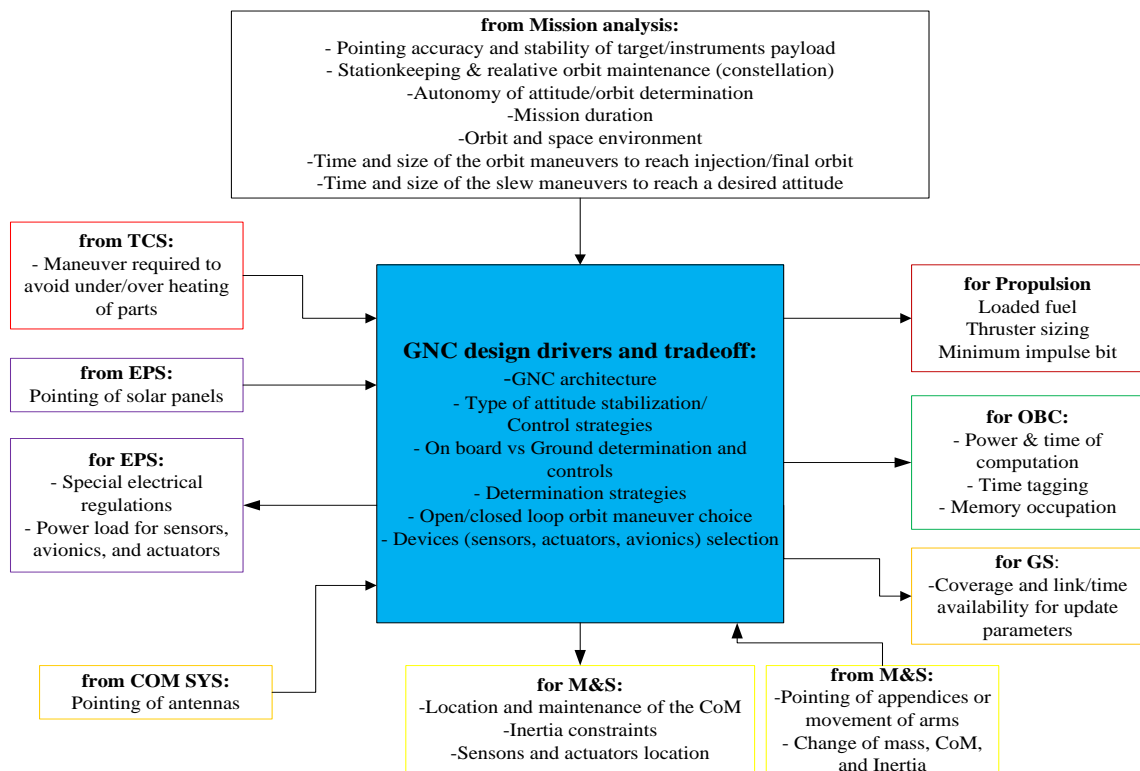


Figure 27: GNC interactions with mission, system and other subsystems elements

At the same time, the GNC may impose constraint to the design of the other subsystems, e.g.:

- Propulsion system in terms of necessary propellant, thrusters sizing, and minimum impulse bit;
- EPS in terms of special regulation and power load for sensors, actuators and avionics;
- On Board Computer (OBC): in terms of computation power and time consumption (e.g. to calculate complex algorithms), time tagging for the geo-reference of the navigation data, and memory occupation (RAM for computation and (P)ROM for firmware and application SW);

- Structure and Mechanism (S&M): in terms of location and maintenance of CoM, inertia constraints, sensors and actuators positioning (e.g. thrusters positioned symmetrically to the principal body axes, IMU/INS placed near the CoM, the star sensors positioned to avoid the over-lighting of Moon and Sun, etc ...);
- Ground Segment: in terms of continuous connection and coverage necessary to update on-board data needed to the GNC activities (e.g. the TLE).

Figure 27 summarizes the interaction among GNC and other subsystems and the whole system.

2.3 GNC design: functions, architectures and features

2.3.1 GNC control modes

This paragraph proposes a general overview on the control modes for the GNC systems.

2.3.1.1 *Orbit acquisition/insertion mode*

The GNC shall lead the spacecraft to achieve the orbit performing the control manoeuvres specified by the mission. The orbit insertion enters after the release of orbiting spacecraft from the launch vehicle and lasts until the final boost that allows reaching the final orbit. In some cases, the End-Of-Life disposal is also a particular case of orbit acquisition: it foresees the reaching of the final disposal orbit and the passivation of the propulsion system, also in degraded configurations due to failures. In this mode the determination of the orbit shall be performed both with on-board equipment and/or with exchange of information with the Ground Segment.

2.3.1.2 *Attitude acquisition mode*

The acquisition mode shall provide the capabilities to determine the angular velocity and the attitude of a spacecraft after the launcher separation and the deployment of appendages (e.g. solar panels or antennas).

2.3.1.3 *Nominal missions mode*

The nominal modes of operations of the GNC strongly depend on the mission's objectives that the entire system shall accomplish. Here after the main examples are listed.

2.3.1.3.1 *Pointing a target*

The spacecraft shall reach a certain attitude and maintain it along the orbit in order to point a target and/or to track a fixed point on Earth or in the space (e.g. the Moon, the Sun, other stars, etc...). Frequently, the attitude is constrained to point a subject to permit that the payload work properly or to reach/maintain/avoid some specific operation condition (e.g. illumination, thermal). Scientific missions are the main example in which "pointing something" are needed for the good accomplishment of the mission.

2.3.1.3.2 *Station keeping*

Station keeping refers to the maintenance of an orbit with a high precision. Station-keeping manoeuvres consist of small (few m/s) adjustments to the velocity of the spacecraft in order to keep the system within some orbit tolerance. They are mainly performed to counteract disturbances as atmospheric drag and gravity effects. Communications missions, navigation missions and formation flying are the main examples of missions in which the stationkeeping is crucial.

2.3.1.3.3 Chasing

Chasing mode refers to rendez-vous activities and consists of a series of open-loop and closed-loop attitude control and orbit manoeuvres of the “chaser” spacecraft in order to mate with another on orbit body. This body could be collaborative (e.g. the ISS), adapting its state parameters to favour the mating activity, or not collaborative (e.g. a debris), whose state parameters and general conditions are not known a priori.

2.3.1.3.4 Boost / Trajectory tracking

The control mode during the ascent phases of a launcher is often called boost. It is characterized by a pre-computed trajectory tracking to pass through the high dynamic pressure region, and the computation of velocity corrections necessary to place the vehicle on the desired trajectory and/or orbit during the ballistic phases.

Trajectory tracking is a more general case of boost in the sense that it refers to any object on orbit or that should reach the orbit following a desired path.

2.3.1.3.5 Disposal

Disposal is the end of the mission life. Disposal manoeuvres lead the spacecraft in a specific orbit with well defined conditions in order to prevent damage to other spacecraft. In general, this manoeuvre is a transfer in “parking” orbits for GEO spacecraft and a safe de-orbiting and re-entry in atmosphere for LEO and, clearly, human mission spacecraft. For objects in LEO the disposal coincides to a controlled or uncontrolled re-entry in the Earth atmosphere: the most critical case is the return in atmosphere for re-usable spacecraft in which the role of GNC (both attitude and trajectory controls) is crucial to save the integrity of the vehicle. At the same way, the re-entry in atmosphere shall be monitored by the GNC also for big expendable object: in fact, their trajectory shall be safe avoiding the crashed landing on populous areas.

2.3.1.4 *Safe mode:*

2.3.1.4.1 Fail Safe

In case of one or more major anomalies, the GNC shall autonomously guarantee the capability to control the attitude (and angular rate) and eventually the orbit to ensure the integrity of the spacecraft from mechanical, thermal, electrical point of view and the maintenance of the communication link.

More in general, the modes transition depends on the defined strategy: it foresees the check of housekeeping telemetry that can touch off events provoking change in the on board software execution flow and hardware configuration. Transition between modes can be caused by ground request (time tagged or not), by on board computer, after checking a transition condition or after a (major) failure detection.

2.3.1.4.2 Anti-Collision mode

To achieve a safe mating in RV missions or to specifically avoid collisions with orbiting object, an anti-collision system (AS) (based mainly on GNC functions and architectures) shall be provided. The activation starts when one or more state parameters of the spacecraft exceed the defined thresholds, detecting a possible collision due to a deviation from the tolerances of nominal behaviour. An escape sequence begins, generally, performing predetermined manoeuvres or computing a new profile for ADCS and ODCS to follow. Sometimes AS is completely independent by GNC but with the same characteristics: it means that AS performs the ADCS function in the critical phase, often using sensors, actuators and computers different from the GNC.

2.3.2 GNC functions

The main high level functions required to GNC are:

- to determine the GNC state variables through measurements and estimation
 - to acquire and process physical information for the attitude determination
 - to acquire data from the sensors (inertial or not inertial)
 - to apply determination algorithms
 - to acquire and process physical information for the orbit determination
 - to acquire and update orbit parameters
 - to propagate the orbit
 - to acquire data from the sensors
 - to estimate the new position in different reference frames
- to perform the attitude and orbit guidance strategies. Note that the attitude is generally determined with on-board sensors, while the orbit can be determined on ground and then sent to the spacecraft via uplink as well as using on-board sensors.
 - to generate profiles of attitude and trajectory for each phases of the mission
 - to determine command to maintain attitude and trajectory profile
 - to acquire and maintain all the attitudes required by the mission operations in the various phases (e.g. LEOP phase, nominal and off normal/degraded situations and emergency manoeuvres)
- to execute the attitude and orbit control strategies
 - to perform the orbit and attitude manoeuvres, according to the mission specifications
 - to provide commands to the actuators
- to process and deliver with a specific frequency the attitude, orbit and related information to other on-board subsystems (or in some cases directly to ground)
 - to gather, format and pass data to other subsystems
- to acquire, validate and execute commands devoted to change the control mode of the GNC
- to guarantee the safe state of the system in every phase of the mission, i.e. avoiding collisions
 - to detect possible collisions
 - to compare status variables within the relative motion frame between two elements
 - to change trajectory
 - to define/load “escape” maneuvers
- to manage emergency and anomaly situations
 - to determine the GNC health status
 - to compare GNC health status with predefined thresholds
 - to detect failures

- to activate and deactivate hardware parts
- to enable and disable software functions
- to support the failures
 - to correct the failures
 - to activate off-nominal procedure
- to determine/acquire time and perform synchronization tasks

2.3.3 GNC architecture

The control structure includes the control system (consisting of all relevant functional behaviours of controller, sensors and actuators) and the controlled system (the space vehicle(s)).

In Figure 28, the general control structure is shown. It is easy to see the feedback control loop: sensors measurements are acquired by the controller, which determine the state variables. According to the control laws defined taking into account the control and the interaction with the environment, control commands are computed and sent to actuators in order to generate torques and/or forces that act on the controlled plant in order to satisfy the control performances. The objective of the closed loop is to compensate for the disturbance that affects the system. The nature and the design of the controller are driven by the nature of the disturbance and by the desired performance (for instance, an integral type drives the effect of a constant disturbance to zero). This issue is a key driver of the control design, which is outside the scope of the present thesis. Controlled system (or plant) can be a satellite, a launcher, a chaser, a rover for exploration, a robot arm, and in general all those space systems that require an active control of their attitude and trajectory/orbit.

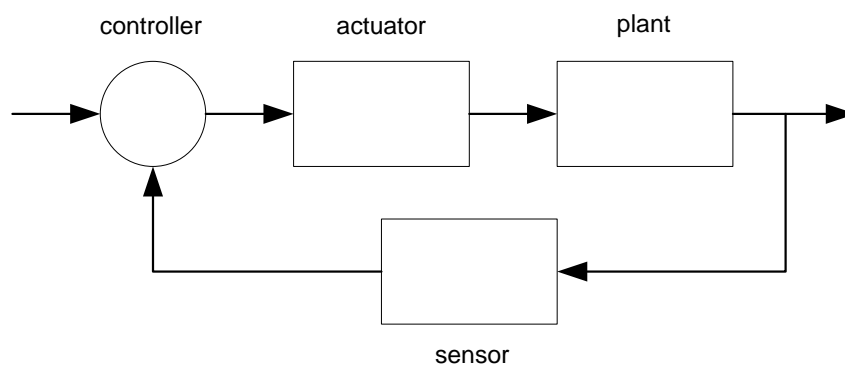


Figure 28: Basic control feedback structure

Figure 29 reports the representation of the feedback control system: the inputs of the controlled system are the commands ($u(t)$) and the external disturbances ($d(t)$) acting on it. The outputs ($y(t)$) of the plant (D) are normally physical parameters (generally, the state variables) of interest for the control performances satisfaction. The outputs are measured (and estimated) by sensors which introduce on the values errors and noise (m). The measured/estimated outputs ($y_m(t)$) are compared with the desired values ($r(t)$) and the computed error (ϵ) is used by the controller (K) to define the commands.

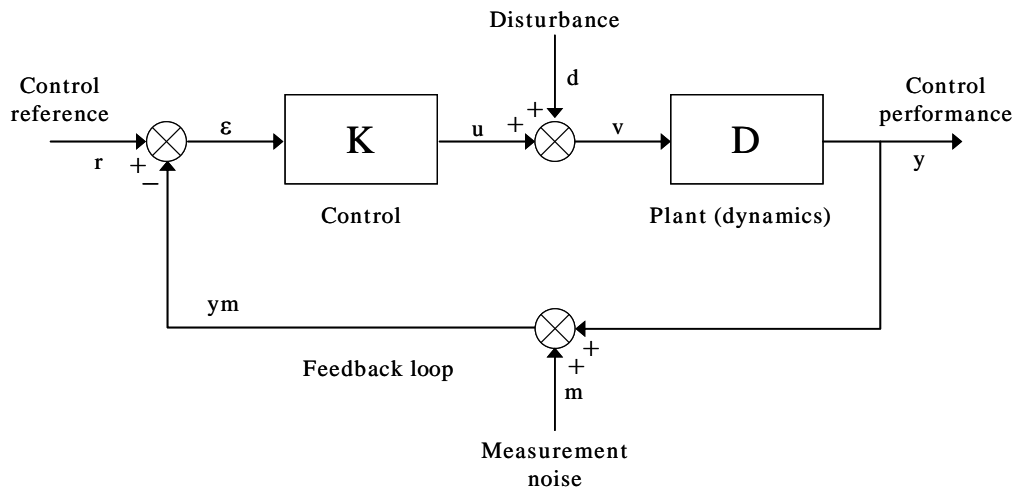


Figure 29: Detailed feedback control system [4]

However, it should be taken into account that in some vehicles or for some phase of the missions, other two types of control shall be implemented:

- **the open loop control:** an open-loop controller, also called a non-feedback controller, is a type of controller that computes its input into a system using only the current state and its model of the system. The main features of the open-loop controller is that it does not use feedback to determine if its output has achieved the desired goal of the input. This means that the system does not observe the output of the processes that it is controlling. Consequently, a pure open-loop system can not engage in machine learning and also can not correct any errors that it could make. It also may not compensate for disturbances in the system.
- **the passive control:** passive stabilization techniques take advantage of basic physical principles and naturally occurring forces (and torques) by designing the spacecraft to enhance the effect of one force while reducing others. In effect, we use the disturbance torques to control the spacecraft, choosing a design to emphasize one and mitigate the others. This kind of control method uses the interaction between the spacecraft and natural phenomena happening in the operational environment (depending on the particular mission), and/or uses the mass characteristics of the spacecraft itself. Major advantage of every passive control system is the ability to obtain a very long spacecraft lifetime, not limited by on-board consumables or, possibly, even by wear and tear on moving parts. Disadvantages are the low pointing accuracy that can be attained and the impossibility to change in response of external events. Moreover, it should be remembered that natural phenomena can change with time, for example during an orbit time. This fact can cause a passively stabilized spacecraft to experience unexpected motions, such as oscillations. In most cases a passive control system include also some processes and devices devoted to the damping of undesired effects.

Thanks to the augmented computer performances and to the miniaturization process that allows increasing the hardware capabilities, controllers have become more accurate and sophisticated, facilitating complex controls based on feedback techniques and more sophisticated and heavy computational algorithms (i.e. LQR, Hinfinity, Neural Network, etc...). Today, the controller family includes analogue on/off logic as well as high complex systems that include digital electronics hardware, multi-scheduled software, elements in the ground segment, logic circuits to detect and correct failures and to change the control strategies. Consequently, the control performances can be very limited (e.g. control a speed of a motor) or very complex and changing with respect to the on

orbit phases. In the latter case, mechanical and electrical configurations of spacecraft, sensors and actuators involved can change from a phase to another.

The allocation of control functions to hardware vs. software vs. human operations, space vs. ground, planning vs. execution (which are essentially independent “dimensions” in implementation) for each mode of a mission are based on trade-offs, which take into account different issues: predictability of the situation, response time, available on-board computer resources, available telecommunications coverage and bandwidth, decision-making complexity, cost of development and operations, and acceptable risk.

2.3.4 GNC in the product life cycle

The control engineering can be considered a sector of SE as highlighted in [5]. As such it can also be broken down into similar engineering activities, that cover the different phase of the product life cycle:

- **Requirements definition**, starting from the mission and system requirements, coherent and appropriate derivation of control requirements from the top level requirements, definition of the control operations requirements, definition of lower component or equipment level requirements (from hardware and software point of view), definition of interface requirements between components and continuous supervision of their status and traceability shall be made.
- **Design** includes definition of the functional control architecture (including the functional interfaces), the derivation of a physical control architecture (including HW, SW and their arrangement) and the controller design able to meeting the control requirements through accurate analyses, trade-offs and taken decisions. The controller design is also supported by control budgets and evaluation of stability and performance (mathematical objects and indexes are defined in paragraph 2.6).
- **Verification and validation** foresee the definition of control verification and validation strategy (including specification of requirements for test environments), the verification of functional performance by analysis on virtual or physical models, through dedicated processes (see the paragraph “In the loop configuration”) that the controlled system meets its control objectives and requirements, and, finally, the in-flight validation of controlled system behaviour.
- **Analysis**, performed at all levels, consists in the selection of adequate analysis tools and methodologies, the requirements verification and (if any) negotiation, the disturbances evaluation, the numerical studies to support the definition of the architecture of control with respect the top level requirements and constraints (i.e. cost, risk and schedule and resources). Moreover, analysis of the results is the base to evaluate and confirm the control system design and the analysis of the performances (also from on orbit data).
- **Integration and control** ensure the integration of the disciplines involved in the control system design throughout all project phases towards the total definition and production of the controlled system. This part covers also the managerial aspects of the control system design, i.e. the monitoring and management of capabilities and resources, the organization and planning of control engineering activities, the definition, management and update of the databases, assessment of control technology and cost effectiveness, the support for the on-orbit operations and maintenance.

Figure 30 underlines the interactions between requirements definition, design, analysis, verification and validation and integration and control for the control process.

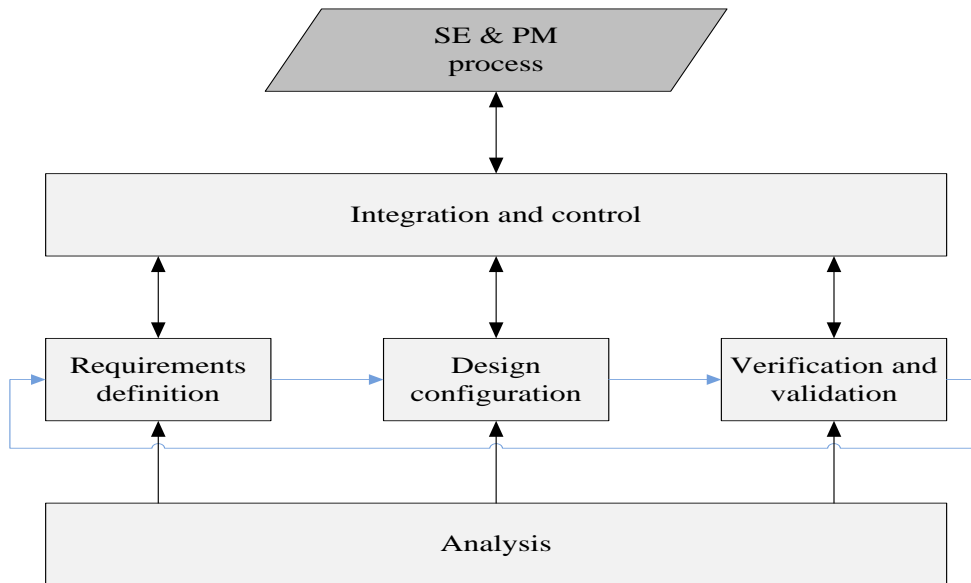


Figure 30: Control Engineering process

All these activities contribute to the proper development of the control system and of its components at various phases of the system product life cycle. The process remains particularly iterative among requirements engineering, design configuration, verification and validation, and analysis. After the functional specification of the control system, hardware, software and operations support items are designed and developed (or procured). Summarizing, as well as SE, the Control Engineering is always actively present, from feasibility studies and requirements definition to verification until, even, on orbit validation.

Table 12, Table 13, Table 14, and Table 15 summarize the Control Engineering activities along the life cycle. [6]

Phase 0/A	Integration and control	Requirements engineering	Analysis	Design and configuration	Verification and validation
Inputs	<ul style="list-style-type: none"> - System development schedule - System development approach and constraints 	<ul style="list-style-type: none"> - System objectives - Mission requirements - System performance requirements 	<ul style="list-style-type: none"> - Controlled system objectives - Preliminary control system requirements 	<ul style="list-style-type: none"> - Control system design concepts of similar space systems 	<ul style="list-style-type: none"> - System verification and validation approach
Tasks	<ul style="list-style-type: none"> - First assessment of control system development cost and schedule - Generation of inputs to the system development approach - Identification of availability and maturity of control technology 	<ul style="list-style-type: none"> - Translate mission and system objectives into preliminary control objectives - Definition of preliminary control requirements - Control system life cycle definition 	<ul style="list-style-type: none"> - Analysis of control requirements feasibility for control system alternatives - Preliminary disturbances evaluation - Preliminary performance assessment - Initial sensitivity analysis - Identification of control system critical aspects 	<ul style="list-style-type: none"> - Establishment and trade-off of control system design concepts - Establishment of control system design baseline (including preliminary FDIR concept) 	<ul style="list-style-type: none"> - Control engineering support for definition of verification and validation concepts - Preliminary definition of control verification and validation methods and strategies
Outputs	<ul style="list-style-type: none"> - Inputs to project and system engineering plan - Inputs to cost estimates and schedule estimates - Inputs to technology development plan 	<ul style="list-style-type: none"> - Inputs to system requirements documentation 	<ul style="list-style-type: none"> - Control system analyses 	<ul style="list-style-type: none"> - Preliminary control system design and analysis report 	<ul style="list-style-type: none"> - Inputs to development and verification planning

Table 12: CE activities in the phases 0+A

Phase B	Integration and control	Requirements engineering	Analysis	Design and configuration	Verification and validation
Inputs	<ul style="list-style-type: none"> - Phase 0/A project planning and cost estimates - Control life cycle Phase 0/A 	<ul style="list-style-type: none"> - System objectives - Mission requirements - Controlled system objectives and requirements 	<ul style="list-style-type: none"> - Phase 0/A simulation models - Phase 0/A control analyses 	<ul style="list-style-type: none"> - Phase 0/A control design 	<ul style="list-style-type: none"> - System verification plan - Phase 0/A control verification plan
Tasks	<ul style="list-style-type: none"> - Update control system inputs to system engineering management plan and cost estimates (including risk management) - Review of the control systems compatibility with the system design and constraints 	<ul style="list-style-type: none"> - Analyse system requirements - Generate controlled system requirements - Allocate controlled system requirements to subsystems and components - Check traceability of control requirements with respect to system requirements 	<ul style="list-style-type: none"> - Analysis of control requirements for sub-systems and components - Disturbances assessment - Controlled system performance analysis - Controlled system sensitivity analysis - Assessment of control technologies for early prototyping 	<ul style="list-style-type: none"> - Definition of control system baseline - Allocation of control system functions to H/W, S/W and human operators (inflight and on ground) - Definition of control system interfaces - Preliminary design of controller (control laws) - Preliminary definition of control related FDIR - Selection of control components and technologies - Establishment of control related budgets and margins 	<ul style="list-style-type: none"> - Prepare controlled system verification plan - Provide inputs to lower level verification plans - Provide inputs to the management plan - Support Phase C/D verification planning
Outputs	<ul style="list-style-type: none"> - Inputs to project and system engineering plan - Inputs to cost estimates and schedule 	<ul style="list-style-type: none"> - Inputs to system or subsystem technical specifications - Inputs to lower level technical specifications - Inputs to requirements database - Inputs to interface control documents 	<ul style="list-style-type: none"> - Controlled system analysis report (including simulation models description) 	<ul style="list-style-type: none"> - Control system design report (incl. design justification) - Preliminary control algorithms specification - Preliminary control system budgets 	<ul style="list-style-type: none"> - Controlled system verification plan - Preliminary controlled system verification report

Table 13: CE activities in the phase B

Phase C/D	Integration and control	Requirements engineering	Analysis	Design and configuration	Verification and validation
Inputs	<ul style="list-style-type: none"> - Phase B project planning and cost estimates - Control life cycle Phase B 	<ul style="list-style-type: none"> - Phase B control objectives and requirements - Phase B control components specifications 	<ul style="list-style-type: none"> - Phase B simulation models - Phase B control analyses 	<ul style="list-style-type: none"> - Phase B control design and design justification 	<ul style="list-style-type: none"> - Phase B controlled system verification plan
Tasks	<ul style="list-style-type: none"> - Support of system engineering and project management (including risk management) - Management of required control system changes - Support of operations - Review of data packages - Support to Phase E/F planning and cost estimate 	<ul style="list-style-type: none"> - Update of specifications - Review and assessment of control requirements changes - Review and assessment of system changes related to control 	<ul style="list-style-type: none"> - Detailed controlled system performance analysis - Update of sensitivity analysis - Support to verification process - Support to inflight verification process definition 	<ul style="list-style-type: none"> - Update of the control design baseline - Finalization of control system functional architecture and interfaces - Detailed design of controllers and optimization of controller parameters - Detailed design of control-related FDIR - Review of control budget and margins analysis 	<ul style="list-style-type: none"> - Coordinate and monitor controlled system and lower level verification plans and activities - Monitor lower level verification acceptance activities - Support and monitor lower level qualification and acceptance tests - Perform controlled system qualification and acceptance tests
Outputs	<ul style="list-style-type: none"> - Updated inputs to project and system engineering plan - Inputs to system database - Inputs to operations handbook or user manual - Updated inputs to cost estimates for Phase E/F 	<ul style="list-style-type: none"> - Updated inputs to system or subsystem technical specifications - Updated inputs to lower level technical specifications - Updated inputs to interface control documents 	<ul style="list-style-type: none"> - Controlled system analysis report - Inputs to the definition of the strategies for the inflight calibration and performance analysis 	<ul style="list-style-type: none"> - Final control system design report - Final control algorithms specification (including control system TM/TC specification) - Final control system budgets 	<ul style="list-style-type: none"> - Controlled system verification report - Inputs to inflight verification plan

Table 14: CE activities in the phases C and D

Phase E/F	Integration and control	Requirements engineering	Analysis	Design and configuration	Verification and validation
Inputs	- System operations planning	- Final system and lower level specifications	- Controlled system requirements - Controlled system on orbit performance data - Strategies for the on orbit performance analysis	- Final control system design report	- On board verification plan
Tasks	- Support of system operations - Management of specified controller changes - Control engineering support to system disposal - Generation of lessons learnt for control engineering	- Comparison of control objectives and requirements with controlled system performance - Clarify control objectives and requirements changes during operation	- Analysis of controlled system operational performance - Analysis of required controller changes	- Update of controller design (in case of required changes)	- Support controlled system operational performance verification - Support system review
Outputs	- Inputs to disposal plan	- New control related operational requirements	- Inputs to controlled system operational performance report - Updated control controlled system analysis report - Inputs to payload data evaluation	- Controller design updates (updated control system design report)	- Inputs to on orbit acceptance report - Inputs to periodic mission reports

Table 15: CE activities in the phases E and F

2.4 Verification and validation of a GNC system

The GNC cannot be fully verified in real conditions before flight. The main reason is that the on board hardware can not be subjected to the real flight conditions and environment on ground. However, the GNC verification process shall follow a complete and careful step-by-step approach that starts from virtual models and ends with the real hardware.

The GNC verification can be developed in the different phases and aspects:

- GNC design and performance verification: it demonstrates that the GNC definition (e.g. modes, architecture, equipment and settings) is compliant with the functional requirements. It includes both analyses (based also on sensitivity and robustness) and simulations.
- GNC hardware/software verification: it verifies the functional GNC behaviour with a representative configuration of hardware/software, interfaces and real-time performances. It concerns the overall functional loop, each part of the GNC (flight hardware and software) has to be individually verified with respect to its own specification in a separate process.
- Entire GNC verification at satellite level: it foresees the same HW/SW verification of GNC integrated in the entire system, rather than having it separated.
- GNC-ground interface verification: it consists of verifying the interactions of the subsystem with the GS (exchange of telemetry and reception of command).
- On orbit verification and validation: it consists of the good working of the system in orbit, confirming the correctness design that will be a milestone in future applications.

This step-by-step verification approach is fully applicable to a program that includes new developments on both hardware and software. The GNC design and performance verification relies on extensive analyses and simulations for a completely new development or for a new family of spacecraft. This requirements verification can be skipped if a lot of hardware units and software functions are reused from a previous development, and when some verification steps performed previously can be considered as applicable to the current program.

The GNC HW and SW verification for a completely new development relies on test benches, which include functionally representative hardware models of flight hardware, while it can use numerical simulation models of some hardware units for a recurring satellite, or for a satellite of an existing family.

2.4.1 Mission definition and feasibility phase

The verification and validation activities for GNC in phase A refer to mission definition analysis and feasibility studies of the whole system (the spacecraft): in particular, mission concepts and requirements have to be validated. These studies concern on:

- the feasibility of trajectories and attitude strategies,
- the feasibility of the manoeuvre w.r.t. the configuration of the system and the preliminary budgets,
- the identification of navigation and control performances in the various mission phases identifying the subsystems operative modes and making assumptions on actuators and sensors,
- the identification of the more relevant external and internal forces, and torques that disturb the vehicle motion.

Analysis of the trajectories, ΔV , manoeuvres duration will yield the basic data required to define the propellant budget and consequently the actuators thrust, the power consumption for magnetic actuators and reaction wheels, the choice of the sensors, the determination and control strategies, and the guidance strategies. Not real-time simulations based on simplified kinematic and dynamics formula

constitute the main tool: key parameters and models characterization derive from historical data, previous projects and experience. Only relevant disturbances are modelled and it may be required to take into account other aspects of the mission or system, e.g. the vehicle geometry is required to define the atmospheric drag.

SCS are the type of simulators needed to perform the simulation activity during this phase.

2.4.2 Design phase

The objectives of the verification and validation in this phase are:

- the effectiveness of algorithms for guidance, navigation and control and, eventually, anti-collision system,
- the achievability of performances requirements for GNC
- the feasibility of the design implementation with the chosen sensors, actuators, data management hardware
- the correctness of environment modelling

The first important task in the design phase is the development of GNC algorithms: they should be verified through closed loop simulations. No detailed models are involved in the loop; sensors, actuators and environment are modelled only to consider their dominant effects: in fact, they are represented by basic functions plus ideal bias or noise, environmental forces and torques are partially represented in relation to the type of mission: e.g. for the simulation of a GEO satellite atmospheric drag is neglected. The expected GNC performance obtained from this simulation becomes the driving factor for the requirements about trajectory/orbit and attitude.

The GNC design and performance verification shall cover all the GNC modes (and related transitions), functions and tasks. The simulator shall be representative and support all these features and issues or it shall allow to add models, items, tools, and facilities.

In the final part of the design phase, the software with the chosen algorithms starts to enter in the simulation loop. More detailed models of the vehicle dynamics, the environmental condition (such as Earth Magnetic Field and radiations effects), the sensors and actuators with disturbances (e.g. taking into account how the sensors output changes w.r.t. changes in the thermal conditions) take place in the simulation architecture. No hardware is still involved as well as real time simulation is not required allowing saving time and/or performing more runs contemporary.

A basic scheme of a closed loop GNC simulation configuration for the design phase is shown in Figure 31. The configuration required is mainly the AIL where the simulation station has in charge any kind of simulation

The GNC design and performance verification shall be performed through theoretical analyses and numerical simulations on the GNC functional simulator. It is useful to include the monitoring of the failures impacting the GNC functions, with their tuning in the GNC design and performance verification.

All analyses and verification tools used in the design phase will typically run in non-real time, since there is no hardware test item requiring a real time environment. Validation of the models of spacecraft and equipment will, at this stage, be limited to a comparison with the evolving design of these items and with elements deriving from previous experiences and projects. Validation of models for dynamic disturbances on orbit and environments will, in most cases, be limited to comparison with data known from previous missions.

Moreover, the GNC design and performance verification shall include a robust analysis covering the nominal variation range specified for the physical data and hardware performances: typical parameters include mass properties, sensor and actuators performances, environmental conditions, orbit uncertainties and drifts.

FES is the simulators family involved in this kind of verification.

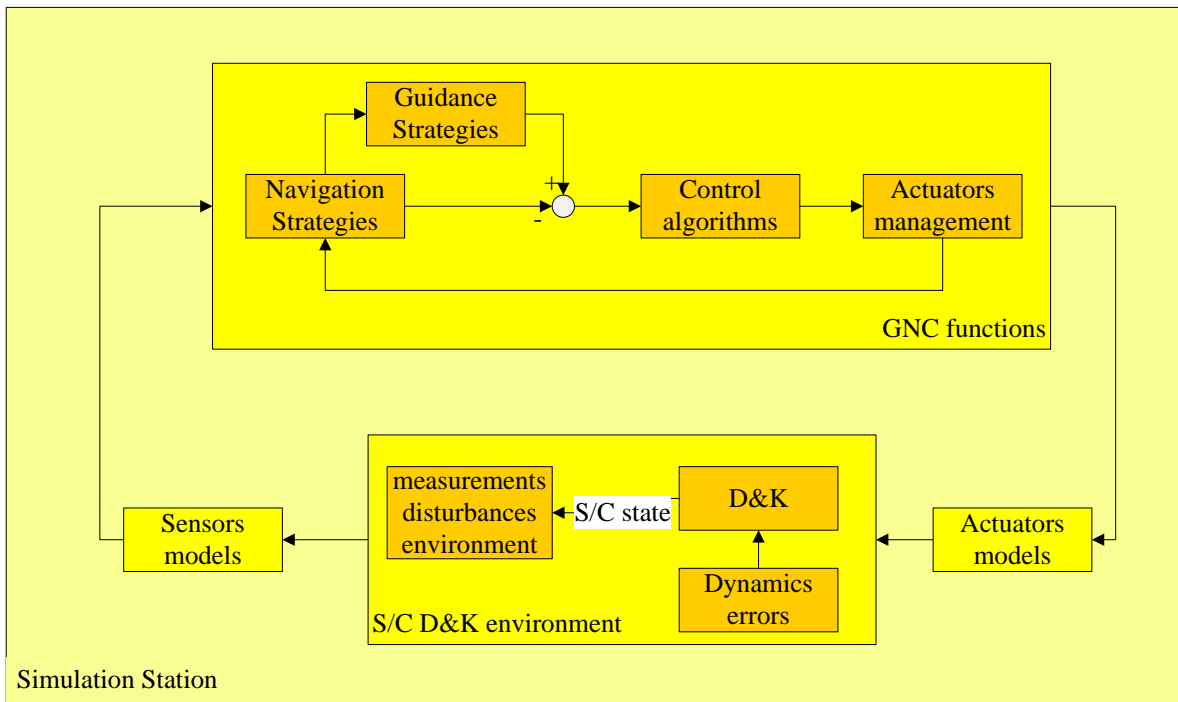


Figure 31: Closed loop GNC simulation configuration during the design phase

2.4.3 Development phase

The verification in this phase looks at verifying:

- Proper function and performance of the complete control system implemented in hardware and software.
- Function and performance of the navigation hardware and software in a realistic measurement environment.
- Proper function of the on-board system together with the remote control functions (i.e. ground segment).

In the development phase, the hardware and software of the system will have to be tested in closed loop in a real time environment with as many as possible real items in the loop. This means that the GNC software shall run in its proper computer hardware and that real sensor hardware shall be connected to it as well as the actuators even if it is generally not possible to include the actuator hardware in closed loop performance tests: in fact, this would require the proper orbital environment for thrusters and spacecraft dynamics to close the loop. To test the system with sensor and controller hardware in the loop, all items involved in the dynamic process, e.g. spacecraft body and actuators must still be represented by virtual models.

During the development phase, the complete on-board software becomes available and its performance has to be proven. The focal tool is a real time simulation, capable to test together the GNC software and the hardware and software for data management. The GNC computer and the interfaces with sensors and actuators, or as an alternative with the mathematical models of these items, are just connected via the on-board data bus. In this way, unexpected behaviour due to their implementation and/or their operations in a different environment w.r.t. the previous simulation will emerge in the test results.

The virtual models for orbital perturbations, for the actuators and their control electronics, for spacecraft dynamics and kinematics, and for sensors and their measurement environment, need to be available with an appropriate level of detail and reliability in order to produce the same closed loop

test result as in the real mission. All models need to be validated to fulfil the requirements, which means being confident that they represent the actual world in a realistic way.

The resulting simulation setup is the main tool for the verification of the function and performance of GNC subsystem, both for the nominal mission and for all predictable non-nominal/contingency situations.

A typical simulation setup for the GNC functions in the development phase is shown in Figure 32.

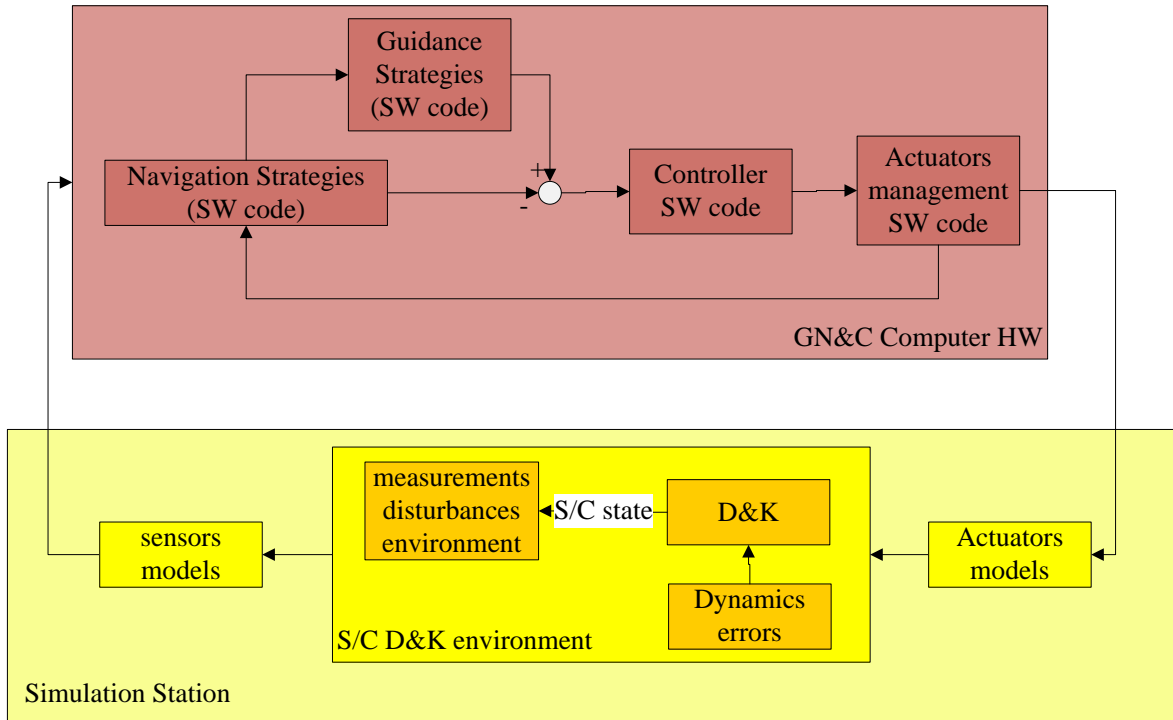


Figure 32: Closed loop GNC simulation configuration at development phase

Special attention should be given to the verification of some equipment, e.g. the sensors: to verify the hardware and software as well as the calibration, test setups with physical stimulation of the sensors are required. First, these could be done with open loop tests (see Figure 33), where the physical stimulation is intended to provide an as realistic as possible measurement environment for the sensors in accordance with the trajectory and attitude motion of the chaser and target spacecraft and the motion of other reference points.

This kind of open loop test setups are often used as the primary means of validation of the mathematical models of the sensors, involved in the SIL simulations.

The disturbance part of the measurement environment models used can be validated only by practical experience in space, either by comparison with data from former missions or by dedicated in-orbit experiments. Once the worst-case situations are known from experience, they can be used in the tests with physical stimulations to validate sensors and navigation function wrt real world conditions.

FVTB and SVF are the simulators on the base of the verification activities during the development of the GNC parts. They can be setup in a desired way, according to the SW and HW availability, time, cost, and schedule.

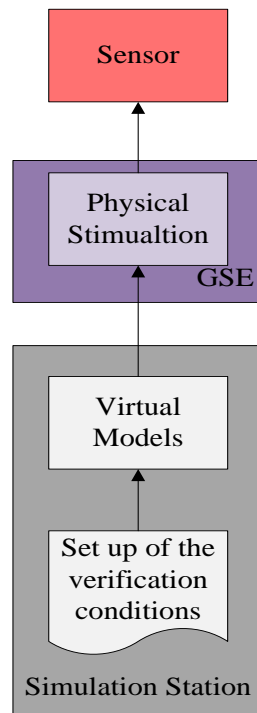


Figure 33: Open Loop GNC stimulation for sensor HW verification

2.4.4 Qualification Phase

The GNC verification for the qualification phase foresees that the system is stimulated because GNC's HW and SW subsystem (stand-alone or integrated in the entire vehicle) shall properly react to external solicitations that reproduces external disturbances or forces as well as the behavior of in the loop components subject of the on orbit environment.

Closed loop tests is needed mainly for the validation on the SIL, CIL and HIL simulation level. Since test preparations of these setups and test runs are complex, long and expensive, tests should be limited to a few particular test cases. Also, such test setups with physical sensor stimulation may not permit the reproduction of contingency situations, either because of physical limitations of the facility or because of operational safety of the test.

One possible arrangement for GNC simulation in this phase is shown in Figure 34.

Sensors can be tested through GSE commanded by the Simulation Station: e.g. according to the computed measurement, the Simulation Station can move accordingly a robotic arm that rotates real system or the parts of it containing a gyroscope. From the actuator point of view, it is quite difficult or impossible to safely guarantee the movement of an object within a facility.

A previous step of this discussed configuration foresees that the real sensors and actuators were out of the loop. Sensors inputs to the GNC computer are directly provided by the GSE (normally different by the previous case): e.g. the output of voltages and currents of a sun sensor can be calculated by the Simulation Station that commands a power supplier able to provide those values directly to the input "sun sensor connector" on the on-board HW.

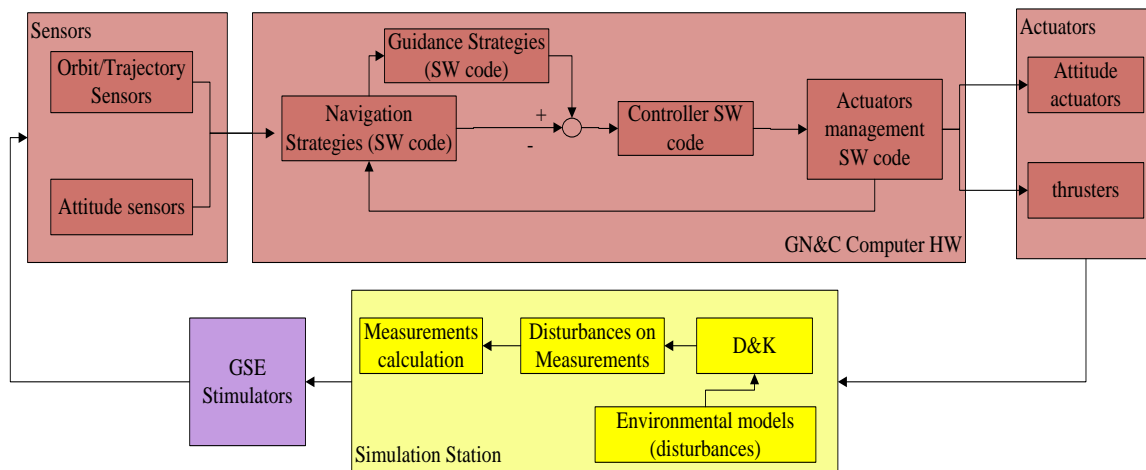


Figure 34: Closed loop GNC simulation configuration at verification stage with real Hardware

2.4.5 Verification at system level

End-to-end GNC tests shall be performed during final verification on the integrated spacecraft. The final GNC functional simulation configuration can be used as a reference to support the end-to-end tests. At this level the GNC shall properly work with the entire system: basic and functional tests are repeated in different environment condition. Referring to Chapter 1, the facilities devoted to perform the test are those derived from the FVTB and SVF and employed according the AIV plan.

2.4.6 GNC-ground interface verification

The interfaces between ground system and GNC with real on-board software are normally verified by test in order to validate the GS. The aim here is to verify the right functional behavior of the GNC submitted to commands sent by the GS during the on orbit life. Accordingly to the mission, different type of command can be transmitted: change of operative control mode, activation/deactivation of GNC's HW or SW parts, and requests of orbit/attitude maneuver or parameter updating. Through this kind of verifications also the correct reception of the GNC on ground can be verified especially if a direct link between the GS and GNC computer has been implemented. GSTS is the type of simulator adoptable for these verifications.

2.4.7 On-orbit verification

Once calibrated, the nominal behaviour of the GNC functional loop shall be verified during the on-orbit commissioning phase of the spacecraft, when the health status is monitored. The on-orbit verification results can be used as a reference for the long-term analysis of the vehicle behaviour and for future project because some parameters can only be verified in orbit.

2.5 Guidance strategies, Navigation algorithms and Control methods.

2.5.1 Attitude Determination

Attitude determination is the process of deriving estimates of actual spacecraft attitude from measurements. Note that we use the term "estimates." Complete determination is not possible; there will always be some error.

ADCS engineers treat two broad categories of attitude measurements. The first, single-axis attitude determination, seeks the orientation of a single spacecraft axis in space (often, but not always, the spin

axis of either a simple spinner or a dual-spin spacecraft). The other, three-axis attitude determination, seeks the complete orientation of the body in inertial space. This may be thought of as single-axis attitude determination plus a rotational, or clock, angle about that axis.

Single-axis attitude determination results when sensors yield an arclength measurement between the sensor boresight and the known reference point. The reference point may be the sun, the Earth nadir position, the moon, or a star. The crucial point is that only an arc-length magnitude is known, rather than a magnitude and direction. Specification of the axis orientation with respect to inertial space then theoretically requires three independent measurements to obtain a sufficient number of parameters for the measurement. In practice, the engineer often selects two independent measurements together with a scheme to choose between the true solution and a false (image) solution caused by the under-specification of parameters. The most common scheme entails using an a priori estimate of the true attitude and choosing the measurement that comes closest to the assumed value.

To effect the three-axis attitude determination requires two vectors that can be measured in the spacecraft body frame and have known values in the inertial reference frame. Examples of such potentially known vectors include, again, the sun, the stars, and the Earth nadir. The key lies in the type of sensor used to effect the measurement rather than in the nature of the reference point. The sensor must measure not merely a simple boresight error, as in single-axis attitude determination, but two angular components of the error vector. The third vector component is known since only unit vectors need be considered in spacecraft attitude control.

2.5.1.1 *Devices*

This paragraph provides a brief overview of the devices and sensors normally involved in the attitude determination providing direct or indirect measurements.

- *Sun sensors* are visible-light detectors which measure one or two angles between their mounting base and incident sunlight. They are popular, accurate and reliable, but require clear fields of view. They can be used as part of the normal attitude determination system, part of the initial acquisition or failure recovery system, or part of an independent solar array orientation system. Since most low-Earth orbits include eclipse periods, Sun-sensor-based attitude determination systems must provide some way of tolerating the regular loss of this data without violating pointing constraints. Sun sensors can be quite accurate (< 0.01 deg) but it is not always possible to take advantage of that feature. Sun sensors are usually mounted near the ends of the vehicle to obtain an unobstructed field of view. Sun sensor accuracy can be limited by structural bending on large spacecraft. Spinning satellites use specially designed Sun sensors that measure the angle of the Sun with respect to the spin axis of the vehicle. The data may be sent to the ground for processing or used in a closed-loop control system on board the vehicle.
- *Star sensors* have evolved rapidly in the past few years, and represent the most common sensor for high-accuracy missions. Star sensors can be scanners or trackers.
 - *Scanners* are used on spinning spacecraft. Stars pass through multiple slits in a scanner's field of view. After several star crossings, we can derive the vehicle's attitude.
 - *Trackers* are employed on 3-axis attitude stabilized spacecraft to track one or more stars to derive 2- or 3-axis attitude information. The most sophisticated units not only track the stars as bright spots, but identify which star pattern they are viewing, and output the sensor's orientation compared to an inertial reference. Putting this software inside the sensor simplifies processing requirements of the remaining attitude control software.

While star sensors excel in accuracy, care is required in their specification and use. For example, the vehicle must be stabilized to some extent before the trackers can determine where they point. This stabilization may require alternate sensors, which can increase total system cost. Also, star sensors are susceptible to being blinded by the Sun, Moon, or even planets, which must be accommodated in their application. Where the mission requires the highest accuracy and justifies a high cost, we use a combination of star trackers and gyros. These two sensors complement each other nicely: the gyros can be used for initial stabilization, and during periods of sun or moon interference in the trackers, while the trackers can be used to provide a high-accuracy, low frequency, external reference unavailable to the gyros. Work continues to improve the sample rate of star trackers and to reduce their radiation sensitivity.

- *Horizon sensors* are infrared devices that detect the contrast between the cold of deep space and the heat of the Earth's atmosphere (about 40 km above the surface in the sensed band). Simple narrow field-of-view fixed-head types (called *pippers* or *horizon crossing indicators*) are used on spinning spacecraft to measure Earth phase and chord angles which, together with orbit and mounting geometry, define two angles to the Earth (nadir) vector. *Scanning horizon sensors* use a rotating mirror or lens to replace (or augment) the spinning spacecraft body. They are often used in pairs for improved performance and redundancy. Some nadir-pointing spacecraft use *staring sensors* which view the entire Earth disk (from GEO) or a portion of the limb (from LEO). The sensor fields of view stay fixed with respect to the spacecraft. This type works best for circular orbits. Horizon sensors provide Earth-relative information directly for Earth-pointing spacecraft, which may simplify onboard processing. The scanning types require clear fields of view for their scan cones (typically 45, 60, or 90 deg, half-angle). Typical accuracies for systems using horizon sensors are 0.1 to 0.25 deg, with some applications approaching 0.03 deg. For the highest accuracy in low-Earth orbit, it is necessary to correct the data for Earth oblateness and seasonal horizon variations.
- *Magnetometers* are simple, reliable, lightweight sensors that measure both the direction and size of the Earth's magnetic field. When compared to the Earth's known field, their output helps us establish the spacecraft's attitude. But their accuracy is not as good as that of star or horizon references. The Earth's field can shift with time and is not known precisely in the first place. To improve accuracy, we often combine their data with data from Sun or horizon sensors. When a vehicle using magnetic torquers passes through magnetic-field reversals during each orbit, we use a magnetometer to control the polarity of the torquer output. The torquers usually must be turned off while the magnetometer is sampled to avoid corrupting the measurement.
- *GPS receivers* are commonly known as high-accuracy navigation devices. Recently, GPS receivers have been used for attitude determination by employing the differential signals from separate antennas on a spacecraft. Such sensors offer the promise of low cost and weight for LEO missions, and are being used in low accuracy applications or as back-up sensors. Development continues to improve their accuracy, which is limited by the separation of the antennas, the ability to resolve small phase differences, the relatively long wavelength, and multipath effects due to reflections off spacecraft components.
- *Gyroscopes* are inertial sensors which measure the speed or angle of rotation from an initial reference, but without any knowledge of an external, absolute reference. We use them in spacecraft for precision attitude sensing when combined with external references such as star or sun sensors, or, for brief periods, for nutation damping or attitude control during thruster firing. Manufacturers use a variety of physical phenomena, from simple spinning wheels (*iron gyros* using ball) to *ring lasers*, *hemispherical resonating surfaces*, and *laser fibre optic*

bundles. The gyro manufacturers, driven by aircraft markets, steadily improve accuracy while reducing size and mass.

Error models for gyroscopes vary with the technology, but characterize the deterioration of attitude knowledge with time (degrees per hour or per square-root of time). When used with an accurate external reference, such as star trackers, gyros can provide smoothing (filling in the measurement gaps between star tracker samples) and higher frequency information (tens to hundreds of hertz), while the star trackers provide the low frequency, absolute orientation information that the gyros lack. Individual gyros provide one or two axes of information, and are often grouped together as an *Inertial Reference Unit*, IRU, for three full axes. IRUs with accelerometers added for position/velocity sensing are called *Inertial Measurement Units*, IMUs.

The attitude determination is based on using a combination of data from sensors and mathematical model to compute the current attitude of the satellite: the problem is that the attitude is usually expressed in form of a rotation matrix (that is the matrix that describes the orientation of the Body frame w.r.t. Inertial frame) or a quaternion, so determination algorithms need at least two measurement vectors. In point of fact, with only one vector the problem is underdetermined, with two it is over-determined so it would be correct to talk of attitude estimation and not of attitude determination. In [9] existing attitude determination methods are categorized as deterministic/statistical solutions and recursive estimation algorithms.

- **deterministic/statistical solutions:** these methods need at least two vector measurements obtained at a single time to determine a three-axis attitude. If a vector measurement is missing the deterministic solutions cannot provide an attitude. Some common deterministic solutions are: TRIAD, q-Method, QUEST and ESOQ [10], [11].
- **recursive estimation algorithms:** the recursive estimation algorithms use both present and past measurements for determining the attitude. The Kalman filters are recursive estimation algorithms utilizing a state-space model of the system [12].

2.5.1.2 *Choice of Attitude determination method*

The choice of attitude determination method is clearly based on what is required for the attitude determination, preferring solutions that are easy to implement and test. Definitely the attitude determination using a deterministic solution is the one that most closely matches the characteristics expressed before. It uses sensor readings at one given point in time (in fact it is also called “single-frame” or “point” method) and does not include dynamics of the system. For example, considering a satellite equipped with sun sensors and magnetometers, the deterministic method can be applied knowing the sensor inputs to the system and by observing if the attitude is estimated in the correct way, while the same cannot be done with the Kalman filter (and similar) because it employs dynamics of the system. On the contrary, the deterministic methods have some shortcomings if compared to the extended Kalman filter: if these weaknesses are acceptable, a deterministic attitude determination method may be suitable.

The deterministic methods require at least two vector measurements to determine the attitude. This is a problem when the satellite is in eclipse or the measured Earth magnetic field vector and the Sun vector are collinear: in this case, the last known attitude may be assumed until the vectors are no longer collinear. In fact, collinearity is assumed not to last longer than 1 or 2 minutes, due to the speed of the satellite. Clearly, when the satellite is in eclipse the Sun vector measurement cannot be used, as well as the stars sensors do not work when the Sun, the Moon or/and other strong sources of lightness stay in the sensors FOV. For all these reasons, the deterministic attitude determination shall be forced to be integrated with a Kalman filter.

Considering the example sensor configuration, it would be possible to estimate the attitude in eclipse by using an Extended Kalman Filter. Attitude determination using an Extended Kalman filter and magnetometer data has been described in [9] and in [13], where data from solar panels are used in addition to magnetometer data, to make the extended Kalman filter converge faster. One difficulty with implementing an extended Kalman filter will be determining a precise inertia matrix. However, if the extended Kalman filter fails, the deterministic solution can be used instead.

2.5.1.3 Deterministic/statistical solutions

Determining the attitude of the satellite is equivalent to finding the rotational matrix \mathbf{R}_{bi} , which describes the orientation of the spacecraft-fixed reference frame, F_b , with respect to a known reference frame (e.g. the inertial reference frame or any frame in which the vector components are known), F_i . Given the generic vector \bar{v} , expressed as v_b and v_i respectively in body and in inertial frame coordinates, the rotation matrix should satisfy the following equation:

$$v_b = \mathbf{R}_{bi} v_i \quad [\text{EQ.1}]$$

If more than two observations are available, a statistical method can be used and it should provide a better estimation of \mathbf{R}_{bi} . In this case there are N unit vectors (with $N \geq 2$) \bar{v}_k , with $k = 1, 2, \dots, N$. For each vector, the rotation matrix is defined as:

$$v_{kb} = \mathbf{R}_{bi} v_{ki} \quad [\text{EQ.2}]$$

The objective is to find a solution that minimizes the overall error for the N vectors. Grace Wahba [31] proposed to find a matrix that minimizes the loss function defined as the sum of the square errors for each vector measurement:

$$J(\mathbf{R}_{bi}) = \frac{1}{2} \sum_{k=1}^N w_k |v_{kb} - \mathbf{R}_{bi} v_{ki}|^2 \quad [\text{EQ.3}]$$

In this expression, J is the loss function to be minimized, k is the counter for the N observations, w_k are non-negative weights, v_{kb} is the k -th observation vector expressed in body frame and v_{ki} is the k -th observation vector expressed in the known (inertial or any other) frame.

2.5.1.3.1 TRIAD algorithm

Given the knowledge of two vectors in both the reference frames, two triads of unit vectors are built and the TRIAD algorithm obtains the direction cosine matrix relating both frames. Each triad is a base of an intermediate auxiliary reference frame, F_t , expressed in the body and inertial frame.

The first base vector is given by:

$$\begin{aligned} \bar{t}_1 &= \bar{u} \\ t_{1b} &= u_b \\ t_{1i} &= u_i \end{aligned} \quad [\text{EQ.4}]$$

The second base vector is given by:

$$\begin{aligned} \bar{t}_2 &= \bar{u} \times \bar{v} \\ t_{2b} &= \frac{u_b \times v_b}{|u_b \times v_b|} \\ t_{2i} &= \frac{u_i \times v_i}{|u_i \times v_i|} \end{aligned} \quad [\text{EQ.5}]$$

Then the third base vector is calculated using the cross product between the vectors calculated in Eq.4 and in [5]:

$$\begin{aligned} \bar{t}_3 &= \bar{t}_1 \times \bar{t}_2 \\ t_{3b} &= t_{1b} \times t_{2b} \\ t_{3i} &= t_{1i} \times t_{2i} \end{aligned} \quad [\text{EQ.6}]$$

Now, given the previous equation, two rotation matrices are obtained:

$$\begin{aligned}\mathbf{R}_{bt} &= [t_{1b} \quad t_{2b} \quad t_{3b}] \\ \mathbf{R}_{it} &= [t_{1i} \quad t_{2i} \quad t_{3i}]\end{aligned}\quad [\text{EQ.7}]$$

The desired attitude matrix is obtained as follows if the matrices are orthogonal:

$$\mathbf{R}_{bi} = \mathbf{R}_{bt}\mathbf{R}_{ti} = \mathbf{R}_{bt}\mathbf{R}_{it}^{-1} = \mathbf{R}_{bt}\mathbf{R}_{it}^T \quad [\text{EQ.8}]$$

2.5.1.3.2 q-Method

The q-Method [32] [33] uses quaternions to solve the attitude determination problem minimizing Wahba's loss function $J(\mathbf{R}_{bi})$. The loss function can be expanded as follows:

$$J(\mathbf{R}_{bi}) = \frac{1}{2} \sum_{k=1}^N w_k (v_{kb} - \mathbf{R}_{bi}v_{ki})^T (v_{kb} - \mathbf{R}_{bi}v_{ki}) = \frac{1}{2} \sum_{k=1}^N w_k (v_{kb}^T v_{kb} + v_{ki}^T v_{ki} - 2v_{kb}^T \mathbf{R}_{bi}v_{ki}) \quad [\text{EQ.9}]$$

As v_{kb} and v_{ki} are unit vectors, $v_{kb}^T v_{kb} = v_{ki}^T v_{ki} = 1$, the loss function simplifies as:

$$J(\mathbf{R}_{bi}) = \sum_{k=1}^N w_k (1 - v_{kb}^T \mathbf{R}_{bi}v_{ki}) \quad [\text{EQ. 10}]$$

The minimization of J is equal to maximization of its first derivative with the opposite sign (gain function, G):

$$\min[J'(\mathbf{R}_{bi})] = \min\left(-\sum_{k=1}^N w_k v_{kb}^T \mathbf{R}_{bi}v_{ki}\right) = \max[G(\mathbf{R}_{bi})] = \max\left(\sum_{k=1}^N w_k v_{kb}^T \mathbf{R}_{bi}v_{ki}\right) \quad [\text{EQ.11}]$$

It is possible to write rotation matrix in form of quaternion in order to obtain this gain function [14]:

$$G(q) = q^T \mathbf{K}q \quad [\text{EQ.12}]$$

where \mathbf{K} is a 4×4 matrix, known as Davenport's matrix, given by:

$$\mathbf{K} = \begin{bmatrix} S - \sigma I & Z \\ Z^T & \sigma \end{bmatrix} \quad [\text{EQ.13}]$$

where

$$\begin{aligned} B &= \sum_{k=1}^N w_k v_{kb} v_{ki}^T \\ S &= B + B^T \\ Z &= [B_{23} - B_{32} \quad B_{31} - B_{13} \quad B_{12} - B_{21}]^T \\ \sigma &= \text{tr}(B) \end{aligned} \quad [\text{EQ.14.a, EQ.14.b, EQ.14.c, EQ.14.d}]$$

q-Method is based on restating the problem of G maximization to obtain an eigenvalue (λ) problem [15]. In this way the largest eigenvalue of \mathbf{K} maximizes the gain function G and the corresponding eigenvector is the least-squares optimal estimate of the attitude.

To maximize the gain function, the derivative with respect to q is considered but since the quaternion elements are not independent, the constraint of equation ($q^T q = 1$) must also be satisfied. Adding the constraint to the gain function with a Lagrange multiplier yields a new gain function:

$$G'(q) = q^T \mathbf{K}q - \lambda q^T q \quad [\text{EQ.15}]$$

Differentiating this gain function, the following equation is obtained:

$$\mathbf{K}q = \lambda q \quad [\text{EQ.16}]$$

that is easily recognizable as an eigenvalue problem:

$$(\lambda I - \mathbf{K})q = 0 \quad [\text{EQ.17}]$$

From the eigenvalue corresponding to the optimal eigenvector (quaternion) that maximize the gain function it is possible to observe that:

$$G(q) = q^T \mathbf{K}q = q^T \lambda q = \lambda q^T q = \lambda \quad [\text{EQ.18}]$$

It is now clear that the largest eigenvalue of \mathbf{K} maximizes the gain function. The eigenvector corresponding to this largest eigenvalue is the least-squares optimal estimate of the attitude. The q-Method solves the eigenvalue/eigenvector problem directly, but it is a very computationally intensive task.

2.5.1.3.3 QUEST

As previously noted, the eigenproblem solution is numerically intensive. This implies that a more efficient way than the direct solution of such problem is needed. One algorithm developed to do this is the QUEST (QUaternion ESTimator) algorithm [34]. Considering equations [EQ.9], [EQ.10], [EQ.18] (with $\lambda = \lambda_{opt}$), it is possible to write:

$$\lambda_{opt} = \sum_{k=1}^N w_k - J \cong \sum_{k=1}^N w_k \quad [\text{EQ.19}]$$

The quaternion is computed by first solving for the Rodriguez parameters \bar{p} (most likely using a numerical method) the following equation:

$$Z = [(\lambda_{opt} + \sigma)\mathbf{1} - S]\bar{p} \quad [\text{EQ.20}]$$

where Z , σ and S are those defined for q-Method and \bar{p} are defined as follows:

$$\bar{p} = \frac{\bar{q}}{q_4} \quad [\text{EQ.21}]$$

Once the Rodriguez parameters are found, the quaternion is calculated by:

$$\bar{q} = \frac{1}{\sqrt{1+\bar{p}^T\bar{p}}} \begin{bmatrix} \bar{p} \\ 1 \end{bmatrix} \quad [\text{EQ.22}]$$

It must be taking into account the singularity when the rotation is π radians, avoidable with a proper method of sequential rotations.

2.5.1.3.4 ESOQ-1 – Estimator of the Optimal Quaternion

[36] illustrates the ESOQ1 method starting from the Davenport's eigenvalue equation [EQ.15] which says that the optimal quaternion is orthogonal to all the columns of the matrix

$$\mathbf{H} \equiv \mathbf{K} - \lambda_{max}\mathbf{I} \quad [\text{EQ.23}]$$

which means that it must be orthogonal to the three-dimensional subspace spanned by the columns of \mathbf{H} (it is a 4×4 matrix but only three of its columns are linearly independent as $\det\mathbf{H} = 0$). The optimal quaternion is conveniently computed as the generalized four-dimensional cross-product of any three columns of this matrix. Examining the adjoint of \mathbf{H} and considering the biggest eigenvalue $\lambda_{max} = \lambda_1$, it can be shown that:

$$adj(\mathbf{H}) = (\lambda_2 - \lambda_{max})(\lambda_3 - \lambda_{max})(\lambda_4 - \lambda_{max})\bar{q}_{opt}\bar{q}_{opt}^T \quad [\text{EQ.24}]$$

where λ_i for $i = 2,3,4$ are the other eigenvalues of \mathbf{H} . Thus \bar{q}_{opt} can be computed by normalizing any non-zero column of $adj(\mathbf{H})$, which is denoted by index k .

Let \mathbf{F} denote the symmetric 3×3 matrix obtained by deleting the k-th row and k- th column from \mathbf{H} , and let f denote the three-component column vector obtained by deleting the k-th element from the k-th column of \mathbf{H} . Then the k-th element of the optimal quaternion is given by:

$$(\bar{q}_{opt})_k = -c \det(\mathbf{F}) \quad [\text{EQ.25}]$$

and the other three elements are:

$$(\bar{q}_{opt})_{1,\dots,k-1,k+1,\dots,4} = c adj(\mathbf{F})f \quad [\text{EQ.26}]$$

where the coefficient c is found by normalizing the quaternion. It is desirable, from a numerical point of view, to let k denote the column with the maximum Euclidean norm, which equation [6] shows to be the column containing the maximum diagonal element of the adjoint.

The original formulation of ESOQ uses the analytic solution of the characteristic equation to obtain the eigenvalues. An alternative way to obtain the maximum eigenvalue λ_{max} , faster and equally accurately, is to iteratively solve the final equation of the FOAM method:

$$0 = \psi(\lambda_{max}) \equiv (\lambda_{max}^2 - \|B\|_F^2)^2 - 8\lambda_{max}\det(\mathbf{B}) - 4\|adj(\mathbf{B})\|_F^2 \quad [\text{EQ.27}]$$

where $\|\cdot\|_F^2$ is the Frobenius norm and \mathbf{B} is defined in q-Method paragraph.

2.5.1.3.5 ESOQ-2 – Second ESTimator of the Optimal Quaternion

[37] illustrates the ESOQ2 algorithm that uses the optimal quaternion written as function of rotation axis \mathbf{e} and angle ϕ . Substituting it into equation [EQ.16] considered for the optimal quaternion, gives:

$$\begin{aligned} [\lambda_{max} - tr(\mathbf{B})] \cos(\phi/2) &= e^T Z \sin(\phi/2) \\ Z \cos(\phi/2) &= \{[\lambda_{max} + tr(\mathbf{B})]\mathbf{I} - \mathbf{S}\}e \sin(\phi/2) \end{aligned} \quad [\text{EQ.28}]$$

Combining these two relations gives:

$$\mathbf{M}e \sin(\phi/2) = 0 \quad [\text{EQ.29}]$$

where

$$\mathbf{M} \equiv [\lambda_{max} - tr(\mathbf{B})]\{[\lambda_{max} + tr(\mathbf{B})]\mathbf{I} - \mathbf{S}\} - ZZ^T = [m_1 \quad m_2 \quad m_3] \quad [\text{EQ.30}]$$

These computations lose numerical significance if $[\lambda_{max} - tr(\mathbf{B})]$ and Z are both close to zero, which would be the case for zero rotation angle. If this happens, a method of sequential rotation can solve the numerical problem.

Equation [EQ.30] says that the rotation axis is a null vector of \mathbf{M} . The columns of $adj(\mathbf{M})$ are the cross products of the columns of \mathbf{M} :

$$adj(\mathbf{M}) = [m_2 \times m_3 \quad m_3 \times m_1 \quad m_1 \times m_2] \quad [\text{EQ.31}]$$

Because \mathbf{M} is singular, all these columns are parallel, and all are parallel to the rotation axis \mathbf{e} . Thus we set:

$$\mathbf{e} = \frac{y}{|y|} \quad [\text{EQ.32}]$$

where y is any column of $adj(\mathbf{M})$ with maximum norm. Because \mathbf{M} is symmetric, it is only necessary to find the maximum diagonal element of its adjoint to determine which column to use. The rotation angle can be found from Equation [28] and substituting it gives:

$$[\lambda_{max} - tr(\mathbf{B})] |y| \cos(\phi/2) = (Z \cdot y) \sin(\phi/2) \quad [\text{EQ.33}]$$

which means that there is some scalar for which

$$\cos(\phi/2) = \eta(Z \cdot y) \quad [\text{EQ.34}]$$

and

$$\sin(\phi/2) = \eta[\lambda_{max} + tr(\mathbf{B})]|y| \quad [\text{EQ.35}]$$

Substituting it into equation of quaternion in rotation axis/angle form, gives:

$$\bar{q} = \frac{1}{\sqrt{[\lambda_{max} - tr(\mathbf{B})]|y|^2 + (Z \cdot y)^2}} \begin{bmatrix} [\lambda_{max} - tr(\mathbf{B})]|y| \\ (Z \cdot y) \end{bmatrix} \quad [\text{EQ.36}]$$

2.5.1.4 Recursive estimation algorithms

The Kalman filter, also known as linear quadratic estimator, is an algorithm which uses a series of measurements observed over time, containing noise (random variations) and other inaccuracies, and produces estimates of unknown variables that tend to be more precise than those that would be based on a single measurement alone. More formally, the Kalman filter operates recursively on streams of noisy input data to produce a statistically optimal estimate of the underlying system state: thanks to its properties, it is an optimal filter for Gaussian errors with zero mean acting on the system, in fact the main assumption of the Kalman filter is that the underlying system is a linear dynamical system and that all error terms and measurements have a Gaussian distribution.

The algorithm works in a two-step process: in the prediction step, the Kalman filter produces estimates of the current state variables, along with their uncertainties. Once the outcome of the next measurement (obviously corrupted with some amount of error, including random noise) is observed, these estimates are updated using a weighted average, with more weight being given to estimates with higher certainty (in fact the weights are calculated from the covariance, a measure of the estimated

uncertainty of the prediction of the system's state). The result of the weighted average is a new state estimate which is between the predicted and measured state, and has a better estimated uncertainty than either alone. Because of the algorithm's recursive nature, it can run in real time using only the present input measurements and the previously calculated state: the entire history of a system's state is not required.

2.5.1.4.1 Discrete Kalman Filter

The Kalman filter [27] addresses the general problem of trying to estimate the state $x \in R^n$ of a discrete-time controlled process that is governed by the linear differential equation

$$x_k = \mathbf{A}_k x_{k-1} + \mathbf{B}u_k + w_k \quad [\text{EQ.37}]$$

with a measurement $z \in R^m$ that is

$$z_k = \mathbf{H}_k x_k + v_k \quad [\text{EQ.38}]$$

The random variables w_k and v_k represent the process and measurement noise. They are assumed to be independent (of each other), white, and with normal probability distributions. The $n \times n$ matrix \mathbf{A} is the state matrix, the $n \times 1$ matrix \mathbf{B} is the control matrix, u_k is the control vector, and the $m \times n$ matrix \mathbf{H} relates the state to the measurement z_k .

As for the continuous case, The Kalman filter can be written as a single equation, however it is most often conceptualized as two distinct phases, "predict" and "update":

- **predict:** the predict phase uses the state estimate from the previous time step to produce an estimate of the state at the current time step. This predicted state estimate is also known as the *a priori* state estimate because, although it is an estimate of the state at the current time step, it does not include observation information from the current time step;
- **update:** in the update phase, the current *a priori* prediction is combined with current observation information to refine the state estimate. This improved estimate is termed the *a posteriori* state estimate.

Typically, the two phases alternate, with the prediction advancing the state until the next scheduled observation, and the update incorporating the observation. Actually, this is not necessary: if an observation is unavailable for some reason, the update may be skipped and multiple prediction steps performed. Likewise, if multiple independent observations are available at the same time, multiple update steps may be performed.

The state of the filter is represented by two variables:

- $\hat{x}_{k|k}$: the *a posteriori* state estimate at time k given observations up to and including at time k
- $\mathbf{P}_{k|k}$: the *a posteriori* error covariance matrix that is a measure of the estimated accuracy of the state estimate.

The predicted (*a priori*) state estimate is:

$$\hat{x}_{k|k-1} = \mathbf{A}_k \hat{x}_{k-1|k-1} + \mathbf{B}u_k + w_k \quad [\text{EQ.39}]$$

with $\hat{x}_{k-1|k-1}$ the *a posteriori* state estimation at time $k - 1$ given the measurements at time $k - 1$.

Predicted (*a priori*) estimate covariance is:

$$\mathbf{P}_{k|k-1} = \mathbf{A}_k \mathbf{P}_{k-1|k-1} \mathbf{A}_k^T + \mathbf{Q}_k \quad [\text{EQ.40}]$$

where $\mathbf{P}_{k-1|k-1}$ is the *a priori* estimate covariance at time $k - 1$ and \mathbf{Q}_k is the covariance of the process noise.

The innovation or measurement residual is:

$$\tilde{y}_k = z_k - \mathbf{H}_k \hat{x}_{k|k-1} \quad [\text{EQ.41}]$$

The innovation (or residual) covariance is:

$$\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k \quad [\text{EQ.42}]$$

where \mathbf{R}_k is the covariance of the observation noise. The optimal Kalman gain is:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1} \quad [\text{EQ.43}]$$

The results are the updated (*a posteriori*) state estimate and the updated (*a posteriori*) covariance estimate:

$$\begin{aligned} \hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \\ \mathbf{P}_{k|k} &= (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} \end{aligned} \quad [\text{EQ.44}]$$

In the actual implementation of the filter, each of the measurement error covariance matrix \mathbf{R}_k and the process noise \mathbf{Q}_k might be measured prior to operation of the filter.

Moreover, it can be noticed that under conditions where \mathbf{Q}_k and \mathbf{R}_k are constant, both the estimation error covariance \mathbf{P}_k and the Kalman gain \mathbf{K}_k will stabilize quickly and then remain constant. If this is the case, these parameters can be pre-computed.

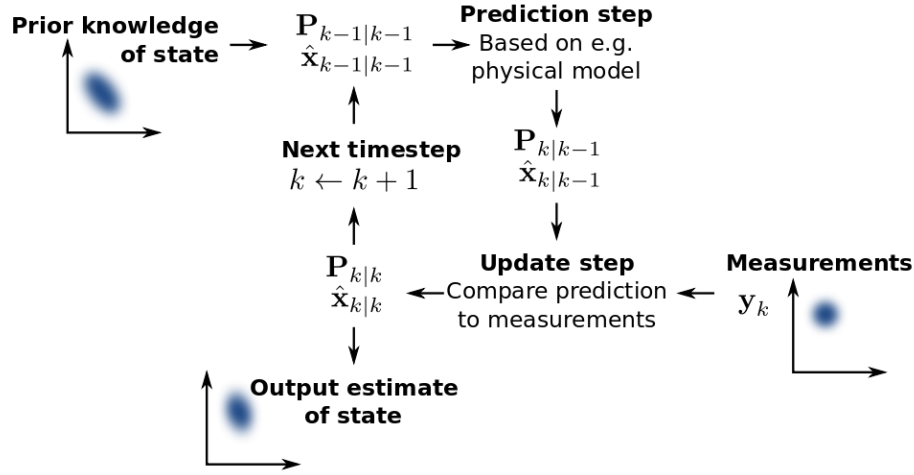


Figure 35: Schematic operation of a Kalman filter

2.5.1.4.2 Extended Kalman Filter

The Extended Kalman Filter (EKF) addresses a problem similar to the one described in the previous paragraph, but it expands it to the non-linear case. It is possible to linearize the estimation around the current estimate using the partial derivatives of the process and measurement functions to compute estimates even if there are non-linear relationships. The process has a state vector $x \in R^n$ now governed by the non-linear differential equation:

$$\dot{x}_k = f(x_{k-1}, u_{k-1}) + w_{k-1} \quad [\text{EQ.45}]$$

with a measurement $z \in R^m$ that is

$$z_k = h(x_k) + v_k \quad [\text{EQ.46}]$$

where the random variables w_k and v_k once again represent the process and measurement noise. In this case the non-linear function $f(\cdot)$ relates the state at time step k to the state at time step $k + 1$ and includes control u_k . The non-linear function $h(\cdot)$ relates the state x_k to the measurement z_k .

The two noises are unknown at each time step but it is possible to approximate the state and measurement vector without them. The predicted (*a priori*) state estimate is:

$$\hat{\mathbf{x}}_{k|k-1} = f(\hat{\mathbf{x}}_{k-1|k-1}, u_{k-1}) \quad [\text{EQ.47}]$$

with $\hat{\mathbf{x}}_{k-1|k-1}$ the *a posteriori* state estimation at time $k - 1$ given the measurements at time $k - 1$.

Predicted (*a priori*) estimate covariance is:

$$\mathbf{P}_{k|k-1} = \mathbf{F}_{k-1} \mathbf{P}_{k-1|k-1} \mathbf{F}_{k-1}^T + \mathbf{Q}_{k-1} \quad [\text{EQ.48}]$$

where \mathbf{Q}_{k-1} is the covariance of the process noise.

- The innovation or measurement residual is: $\tilde{\mathbf{y}}_k = z_k - h(\hat{\mathbf{x}}_{k|k-1})$ [EQ.49]

- The innovation (or residual) covariance is: $\mathbf{S}_k = \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{R}_k$ [EQ.50]

where \mathbf{R}_k is the covariance of the observation noise. The near optimal Kalman gain is:

$$\mathbf{K}_k = \mathbf{P}_{k|k-1} \mathbf{H}_k^T \mathbf{S}_k^{-1} \quad [\text{EQ.51}]$$

The results are the updated (*a posteriori*) state estimate and the updated (*a posteriori*) covariance estimate:

$$\begin{aligned} \hat{\mathbf{x}}_{k|k} &= \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k \tilde{\mathbf{y}}_k \\ \mathbf{P}_{k|k} &= (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \mathbf{P}_{k|k-1} \end{aligned} \quad [\text{EQ.52}]$$

where the state transition and observation matrices are defined to be the following Jacobians:

$$\begin{aligned} \mathbf{F}_{k-1} &= \left. \frac{\partial f}{\partial \mathbf{x}} \right|_{\hat{\mathbf{x}}_{k-1|k-1}, u_{k-1}} \\ \mathbf{H}_k &= \left. \frac{\partial h}{\partial \mathbf{x}} \right|_{\hat{\mathbf{x}}_{k|k-1}} \end{aligned} \quad [\text{EQ.53}]$$

An important feature of the EKF is that the Jacobian \mathbf{H}_k in the equation for the Kalman gain \mathbf{K}_k correctly propagates or magnifies only the relevant component of the measurement information.

A more general formulation does not consider that the noises are negligible, so the predicted (*a priori*) estimate covariance and the innovation (or residual) covariance are:

$$\begin{aligned} \mathbf{P}_{k|k-1} &= \mathbf{F}_{k-1} \mathbf{P}_{k-1|k-1} \mathbf{F}_{k-1}^T + \mathbf{L}_{k-1} \mathbf{Q}_{k-1} \mathbf{L}_{k-1}^T \\ \mathbf{S}_k &= \mathbf{H}_k \mathbf{P}_{k|k-1} \mathbf{H}_k^T + \mathbf{M}_k \mathbf{R}_k \mathbf{M}_k^T \end{aligned} \quad [\text{EQ.54}]$$

where the matrices \mathbf{L}_{k-1} and \mathbf{M}_k are Jacobian matrices:

$$\begin{aligned} \mathbf{L}_{k-1} &= \left. \frac{\partial f}{\partial \mathbf{w}} \right|_{\hat{\mathbf{x}}_{k-1|k-1}, u_{k-1}} \\ \mathbf{M}_k &= \left. \frac{\partial h}{\partial \mathbf{v}} \right|_{\hat{\mathbf{x}}_{k|k-1}} \end{aligned} \quad [\text{EQ.55}]$$

This formulation is implemented the same way as that one seen above.

2.5.2 Orbit determination

There are various ways to determine the orbit with different accuracies:

- **real-time orbit determination:** it provides the best estimate of where a satellite is at the present time and may be important for spacecraft and payload operations, such as accurate pointing at some target;
- **definitive orbit determination:** it is the best estimate of the satellite position and orbital elements at some earlier time, it is done after gathering and processing all relevant observations;
- **orbit propagation:** it refers to integrating the equations of motion to determine where a satellite will be at some other time. Usually orbit propagation refers to looking ahead in time from when the data was taken and is used either for planning or operations. Occasionally orbits will be propagated backward in time, either to determine where a satellite was in the past or to look at historical astronomical observations in the case of comets or planets.

Traditionally, ground stations from around the world provide tracking data to a mission-operations centre. When all data is available, definitive orbit determination provides the best estimate of the orbit. This is used to process the payload data for science or observation missions. The best estimate of the orbit is then propagated forward for real-time operations (such as star catalogue selection or manoeuvre timing) and further forward for mission planning.

In 1983 NASA launched the first *Tracking and Data Relay Satellite*, *TDRS*, to begin replacing the worldwide ground tracking network. TDRS provides the same functions as the traditional ground-station network. As the name implies, it tracks low-Earth orbiting satellites and relays data between the satellite and the TDRS ground station in White Sands, NM.

GPS, GLONASS, and other more autonomous systems are also becoming operational, so orbit determination for future systems will differ significantly from what it has been in the past.

The observations used for orbit determination can be obtained by tracking from the ground, tracking from space, or from autonomous or semi-autonomous systems on the spacecraft. Each of these approaches will be described.

- *Ground tracking* is the traditional way to obtain data for orbit determination. We either track the spacecraft's telemetry signals or use radar tracking from a site not associated with the spacecraft. In both cases, the principal data used for orbit determination are *range* and *range rate*, that is, the distance from the ground station to the satellite and the satellite's line-of-sight velocity during the overhead pass. Angular measurements are also available at times but are typically far less accurate than range or range-rate measurements. Accurate orbit determination using ground-station data ordinarily requires a number of passes. We may accumulate data from multiple passes over a single ground station, or may receive data at a central location from multiple ground stations around the world. In either case, data from a number of passes goes to one place for processing through a large system such as GTDS (Goddard Trajectory Determination System - NASA). Ground-based systems necessarily operate on historical data and therefore will use propagated orbits for real-time operations and mission planning. Accuracies achievable with ground-based tracking vary with a spacecraft's orbit and the accuracy and amount of data. However, accuracies typically range from several kilometres for low-Earth orbits to approximately 50 km for geosynchronous orbit.
- The *Tracking and Data Relay Satellite, TDRS*, has now replaced NASA's worldwide ground tracking network. A major advantage of this system is that the two operational TDRS satellites can provide tracking data coverage for 85% to 100% of most low-Earth orbits. (TDRS does not work for satellites in geosynchronous orbit). The system collects mostly range and range-rate data from the TDRS satellite to the satellite being tracked. Angular information is available, but is much less accurate than the range and range-rate data. If atmospheric drag effects on a satellite are small, TDRS can achieve accuracies of about 50 m. This is considerably better than most ground-tracking systems. Another way to track from space is to use satellite-to-satellite or crosslink tracking.

Manufacturers have developed a number of autonomous navigation systems for spacecraft. Determining the orbit on board is technically easy with the advent of advanced spacecraft computers and higher-order languages. The principal problem is to provide orbit determination that is reliable, robust, and economical in terms of both cost and weight.

Autonomous navigation is inherently real-time. Thus, definitive orbit solutions and payload data are available simultaneously. Moreover, measurements can be less accurate than those for systems that work on old data, because solutions propagated forward in time lose accuracy. With real-time systems highly accurate orbit propagation is less critical, although we will still need some forward propagation for prediction and planning.

- GPS receivers use signals from four different GPS satellites to solve simultaneously for the three components of the observer's position and the time. This can be done several times, providing position and velocity data which determines the orbit elements. The GPS constellation is in a 12 hour orbit at approximately half-geosynchronous altitude. The system provides a moderate accuracy signal (50 m - 100 m) for general navigation and a high-accuracy coded signal (15 m) for military applications. Commercial GPS receivers are now available for spacecraft, and are gaining in popularity in low-Earth orbit.
- A number of proposals have been made for using satellite crosslinks to provide orbit determination. This is of interest because it can be done with crosslink equipment used for inter-satellite communication, and, therefore, requires minimal additional hardware. Crosslink tracking tends not to be implemented because of several practical problems. One problem is that satellite-to-satellite tracking provides only the relative positions of the satellites in the

constellation. This means that if the absolute position is needed for any purpose, then an additional system must be provided to establish the orbit relative to the Earth's surface. A second problem is that the satellites become interdependent, so satellite-to-satellite tracking may not work well for the first satellites or may degrade if a satellite stops working. Therefore, an alternative system not based on satellite-to-satellite tracking is required. If additional systems must be provided, there is less benefit from the satellite-to-satellite tracking.

- A number of approaches for orbit and attitude determination have been proposed, based on the interaction of starlight with the Earth's atmosphere (*Stellar Refraction Systems*). Specifically, as stars approach the edge of the Earth a
- s seen from the spacecraft, refraction will cause their position relative to other stars to shift, producing an effect which can be measured with considerable accuracy. Theoretical accuracies for such systems are projected to be in the vicinity of 100 m. However, none of these systems has been fully developed for flight as yet. The combination of *Earth and star sensing* works similarly to sensing the Earth, Sun, and Moon. The direction and distance to the Earth are sensed relative to the inertial frame of the fixed stars. This is then used to directly determine the direction and distance to the spacecraft. The Earth and stars are available nearly continuously in any Earth orbit and star identification is becoming less of a problem with the introduction of substantially better computers for space use.
- *Landmark tracking* has also been proposed for orbit determination. This has been established as feasible by using data returned from satellite payloads. However, it has not been used as a normal method for satellite navigation, due in part to the difficulty of establishing automatic, unambiguous identification of landmarks to ensure that tracking accuracy can be maintained in the presence of adverse weather or poor seeing conditions.

2.5.3 Guidance strategies

This paragraph presents the most common guidance functions for the three types of considered vehicles: a (small) satellite, a chaser, and a launcher, [8], [1], [3], and alii.

2.5.3.1 Satellite

Orbital phase	Guidance function
Detumbling / damping phase	<ul style="list-style-type: none"> • To damp the satellite angular velocity after the release from the launcher • To reduce the satellite angular velocity wrt an inertial frame under a pre-defined threshold • To stabilize the satellite wrt a specified attitude for the start of the orbit transfer maneuver.
Transfer to final orbit	<ul style="list-style-type: none"> • To verify if the initial point (with a certain tolerance) for orbit transfer is reached • To follow a pre-determined trajectory • To verify the final point reaching • To verify, if any, the reaching of middle point along the trajectory
Pointing phase	<ul style="list-style-type: none"> • To determine the desired pointing • To compute closed-loop manoeuvres to reduce/eliminate the error between the actual attitude and the desired attitude • To maintain the final, desired attitude, computing commands to counteract the disturbance torques

Station-keeping phase	<ul style="list-style-type: none"> To maintain the orbit position w.r.t. an inertial frame, computing commands that neutralize the disturbance forces.
Disposal phase	<ul style="list-style-type: none"> To verify if the initial point (with a certain tolerance) for disposal orbit transfer is reached To follow a pre-determined trajectory To verify the final point reaching (only if the disposal is on-orbit)

Table 16: Guidance strategies for a satellite

2.5.3.2 Rendezvous and docking

Orbital phase	Functions
Phasing	<ul style="list-style-type: none"> To reduce the phase angle between chaser and target through one of the following manoeuvre types: forward/backward phasing, circular/elliptic phasing orbits, change of orbit height (circular orbit), change of apogee/perigee height (elliptic orbits) To perform corrections to reduce/eliminate the injection errors for RAAN and inclination To reach the “initial aim point” and the “trajectory gate”(ranging position and velocity)
Far RV	<ul style="list-style-type: none"> To acquire target orbit To reduce the approach velocity
Close RV	<ul style="list-style-type: none"> To reduce the distance from the target To reach conditions necessary to enter in the final corridor (along V-bar or R-bar)
Final to approach	<ul style="list-style-type: none"> To reach predefined final conditions for the mating in terms of relative distance and attitude between chaser and target To follow the motion of the docking port
Mating	<ul style="list-style-type: none"> To deliver the chaser capture interface into the reception range of the target maintaining specific conditions in terms of approach velocity, lateral alignment, angular alignment, lateral and angular rates (for docking) or position and attitude accuracy, residual linear and angular rates (for berthing)
Departure and de-orbiting	<ul style="list-style-type: none"> To compute the departure, not-return trajectory (along V-bar or R-bar) To compute the deviation from this trajectory w.r.t. the orbit and the attitude

Table 17: Guidance strategies for chasers spacecraft

2.5.3.3 Launchers

Orbital phase	Guidance functions
Boost – ascent phase	<p>To use pre-computed early boost trajectory to pass through region of high dynamic pressure</p> <p>To compute, during successive boost, velocity corrections necessary to place vehicle on desired trajectory or orbit</p>

Orbit – ballistic phase	To compute deviations from stable attitude or required attitude change To compute deviations from required orbit or changes needed to obtain new orbit
Re-entry	To compute deviations from re-entry attitude To compute desired direction and magnitude of available aerodynamic forces to shape re-entry trajectory

Table 18: Basic guidance strategies for launch vehicles

2.5.4 Control: methods and techniques

2.5.4.1 Attitude control

The Table 19 shows the main attitude control methods and summarizes their features.

Type	Pointing options	Maneuverability	Typical Accuracy	Lifetime limits
Gravity gradient	Earth local vertical only	Very very limited	± 10 degree (2 axes)	no
Gravity gradient and Momentum wheel	Earth local vertical only	Very limited	± 10 degree (3 axes)	Life of the wheel
Passive magnetic	North/south only	Very very limited	± 10 degree (2 axes)	no
Pure Spin stabilization	Inertially fixed any direction Repoint with precession maneuvers	High propellant usage to move stiff momentum vector	± 0.1 to ± 1 degrees (proportional to the spin rate), 2 axes	Thrusters propellant (if applied)
Dual spin stabilization	Limited only by articulation on despun platform	High propellant usage to move stiff momentum vector; Despun platform constrained by its geometry	± 0.1 to ± 1 degrees (proportional to the spin rate), 2 axes	and de-spin Thrusters propellant (if applied) bearings
Bias momentum (1 wheel)	Best solution for local horizontal pointing	Momentum vector of the wheel prefers to stay normal to orbit plane, constraining yaw maneuver	± 0.1 to ± 1 degrees	Propellant and life of avionics and wheel bearings
Zero momentum (thrusters only)	No	No constraints	± 0.1 to ± 5 degrees	Propellant
Zero momentum (three wheels)	No	No constraints	± 0.001 to ± 1 degrees	Propellant (if applied) and life of avionics and wheel bearings
Zero momentum CMG	No	No constraints	± 0.001 to ± 1 degrees	Propellant, life of avionics and wheel bearings
Active magnetic	No	One axis pointing is lost in the some singularity points	± 0.1 to ± 5 degrees	Life of avionics

Table 19: Main attitude control methods [1]

The complete description of the cited control methods and functions is in [1], [2], [16], [8], and [17].

2.5.4.2 *Orbit and trajectory control*

The trajectory control refers to the types of orbital maneuvers performed in open or closed loop, generally using small or large thrusters as actuators:

- Free drift motion: no thruster are involved – “natural orbiting” from initial condition
 - Motion on a coplanar orbit at different altitude
 - Release from a station along R-bar (for RV missions only)
 - Release from a station along H-bar (for RV missions only)
- Impulsive manoeuvres: they foresee instant change of velocity, similar to boost manoeuvres
 - Thrust in an orbital direction
 - Hohmann transfers
 - Thrust in a radial direction
 - Radial impulse transfer along V-bar (for RV missions only)
 - Radial impulsive fly-around (for RV missions only)
 - Thrust in out-of-plane direction (orbit plane correction)
 - Lambert transfer
- Continuous thrust: continuous application of control forces (open or closed loop) along the trajectory
 - Straight line along the velocity vector approach
 - with constant velocity
 - with a velocity profile
 - Straight line along the radial vector (R-bar, mainly for RV mission)
 - with constant velocity
 - with a velocity profile
- Station- keeping on a position outside the target orbit
 - below /above the target orbit
 - out-of-plane position
- Transfer by continuous x-thrust
 - quasi-impulsive x-thrust
 - continuous x-thrust transfer to a different altitude
- Transfer by continuous z-thrust
 - quasi impulsive z-thrust
 - continuous thrust transfer along velocity vector
 - continuous thrust in y-direction
- Forced motion circular fly-around

2.5.4.3 Satellite

Orbital phase	Control methods
Detumbling / damping phase	Attitude: <ul style="list-style-type: none"> • Rate damping + desired attitude acquisition • Gravity gradient + damper • Three-axis stabilization through thrusters, reaction wheels and/or magnetic torque. Orbit: N/A
Transfer to final orbit	Attitude: is related to the control of the trajectory Orbit and trajectory: moderate orbit change or major orbit change
Pointing phase	Attitude: <ul style="list-style-type: none"> • Gravity gradient (2 axis only) for a low accuracy, very limited manoeuvrability, no life time problem • Passive magnetic • Pure spin stabilization • Dual spin • Bias momentum (1 wheel) • Three axes stabilization – active magnetic • Zero momentum (thruster only) • Zero momentum (three wheels) • Zero momentum (CGM) Orbit/trajectory: N/A
Station-keeping phase	Attitude: can be disturbed by the not perfect actuation of the thruster to maintain the orbit. So it could be useful to maintain the attitude using one of the methods presented for the <i>pointing phase</i> . Orbit/Trajectory: orbit maintenance through vernier thrusters
Disposal phase	Attitude: three-axes active control to reach and maintain an attitude that helps the orbit/trajectory manoeuvre(s)

Table 20: Control strategies and related techniques for satellites

2.5.4.4 RVD/B

In mission of RVD/B, the performance requirements for the reduction of the trajectory errors increase with decreasing range to the target. For example, for manoeuvre at 30-50 Km from the target the necessary increments ΔV per boost are smaller w.r.t. the orbit boost during the phasing but they are higher w.r.t. the manoeuvre in mating or close approach. The last consideration highlights that different actuators have to be used according to the phases: large thrusters are needed for phasing and far RV manoeuvre, vernier thruster as well as CGM or reaction wheels for the attitude control shall be used in the latter phases before the mating and, immediately after the departure. The final manoeuvres to re-entry or change the orbit still require the larger thrusters.

Orbital phase	Control features
Phasing and far RV	<ul style="list-style-type: none"> • Open loop boost manoeuvres: large thrusters are fired for a duration calculated from the expected acceleration and the required ΔV. Sometimes middle-course open loop manoeuvres should be performed in order to correct too high error due to the previous open loop manoeuvres that can carry out a too low accuracy • Rarely, trajectory that heavily changes the orbit of the spacecraft can be performed with a low but continuous acceleration during the transfer. Closed loop with high bandwidth maneuver are performed using small

	thrusters for long time. The high bandwidth allows controlling “continuously” both the trajectory and the attitude that can deviate for long firings of the thrusters, uncertainties, difference in the thruster levels, etc...
Close RV	<ul style="list-style-type: none"> • Control is sufficiently accurate to satisfy constraints due to optical sensors range and the reaching of the final corridor. • Closed loop manoeuvre are surely indicated taking into account that the main issue is to follow a pre-defined trajectory and correct the deviation from the desired state. • A high bandwidth is required to track the profile of velocities, relative position and attitude, maximizing the performances in terms of consumption, time to approach, accuracy. • Good controllers can be LQR and LQG and a well trained Neural Network
Final to approach	<ul style="list-style-type: none"> • Closed loop control with the higher possible bandwidth has to be considered because high capability to react wrt. disturbances and changing reference is required. • Small vernier thrusters can be used but they influence both the translational and the rotational mode, so they should be coupled with CGM or high-speed reaction wheels to guarantee the optimal alignment with the docking port or the berthing element. • Elegant design can require H_∞ (in general, robust controllers) because they have high capability to recover the desired state in presence of larger uncertainties and disturbances.
Mating	<ul style="list-style-type: none"> • Very small actuations to finalize the rendezvous can be made but, in general, thrusters cannot fire in the last meters before the docking/berthing. • Electric thruster for trajectory control in this phase should be evaluated as well as RW or CGM for the attitude control. • In some case chaser shall be completely switched off before the mating: it means that no attitude and trajectory control are permitted in the last 2-3 meters.
Departure and de-orbiting	<ul style="list-style-type: none"> • Closed-loop control is made to damp the angular velocity after the release (as if it was a detumbling manoeuvre). • When the attitude is stabilized, the thrusters fire in order to perform the computed escape manoeuvre and to reach a new orbit, in general for the re-entry on the Earth. • LQR is a good controller for this space as well as the robust controls

Table 21: Control strategies and related techniques for RVD/B

2.5.4.5 Launcher and Re-entry vehicle

Orbital phase	Control function
Boost	<ul style="list-style-type: none"> To maintain attitude and attitude rates within safe limits during region of high dynamic pressure To direct thrust vector to provide velocity changes commanded by guidance Normally, robust and/or optimal control techniques are chosen in order to maximize the system capability to track a reference (the pre-determined trajectory) and react to the disturbance.
Orbit	<ul style="list-style-type: none"> To change/stabilize attitude via thrusters or other actuators Thrusters to move to a new orbit
Re-entry	<ul style="list-style-type: none"> To change/stabilize attitude via thrusters or other actuators Thrusters to move to a new orbit

Table 22: Control strategies and related techniques for launcher and re-entry vehicle

2.6 GNC performance

The performance elements of a control system refer to [18]:

- **extrinsic performance elements:** performance in steady state (converged) conditions, expressed in time or frequency domains; performance wrt transient events. They depend on the interaction between the system and the external conditions. Consequently, assessing this kind of performances carries out to quantify the system environment (e.g. control reference/desired attitude/orbit/trajectory, measurement noise level, disturbance amplitude and size).
- **intrinsic performance elements,** mainly – but not exclusively – for closed-loop controlled systems, are focused on the properties of the feedback loops (e.g. stability, stability margins, robustness, noise rejection). They do not depend on the external conditions because they are exclusively determined by the goodness of the design (e.g. reduced overshoot).

Table 23 lists the extrinsic and intrinsic performances.

In steady state, it is of interest the capability of the controlled system to properly achieve the desired condition (fixed, e.g. pointing the Earth, or varying along time, e.g. chasing another on-orbit object) against internal and external disturbances, signals and measurements noise, computation delays, and hardware or software failures. Possible requirements examples are:

- Maximum absolute pointing error shall be less than 0.05 rad in converged conditions;
- Maximum relative velocity between chaser and target for the final to approach phase shall be less than 1 m/s;
- RMS value of the angular rate measurement error shall be less than 0.1°/s.

	Intrinsic	Extrinsic
Steady state properties	Measurement noise transmission, rejection of external noise disturbance	Absolute pointing error, pointing stability, absolute measurement error
Transient properties	Response time, settling time, damping, etc...	Overshoot (e.g. when an actuator is activated), “tranquilization” time (e.g. when an actuator is deactivated), etc...
Other properties	Stability, stability margins robustness	Fuel/electrical power consumption, average illumination

Table 23: Extrinsic and intrinsic performances

The performances during transient situation (e.g. a control mode change) are generally handled by a different set of requirements because they refer to short and particular conditions. The needs in transient condition can be explained in terms of:

- *Overshoot*: the maximum overshoot of the attitude wrt x,y,z axes shall be less than 1° ;
- *“Tranquilization time”*: the control system shall recover the desired pointing within 30 seconds after worst disturbance occurrence case.

Other requirements derive from general considerations:

- final position accuracy: e.g. the S/C shall reach the final orbit with an accuracy TBC meters
- total consumption during a given phase: Maximum fuel consumption of a chaser spacecraft to pass from injection orbit and target orbit shall be less than 100 Kg;
- total duration of a given phase: the satellite detumbling shall end before one day from the start of the on orbit mission;

Clearly, requirements are defined at each level (system, subsystem and component level). For example, they can refer:

- Number of thrusters firings;
- Average illumination;
- Number of expected desaturation for reaction wheels.

2.6.1 Performance indicators

In order to formalize in the best possible way performance requirements of a GNC system, a set of mathematical indicators shall be defined. Also in this case a general division can be made between extrinsic and intrinsic indicators.

- *Extrinsic indicator*: aim at qualifying the end-to-end behaviour of the control system submitted to environment and measurement disturbances. The definition of these indicators starts from the state variables and parameters of interests for the performance: attitude (Euler angle, quaternion), angular velocity, position, linear velocity, centre of mass position, mass. After that, the error functions and the operator applied should be defined: they are mathematical formula (algebraic differences, distance, angular distance, norm of vectors difference) which quantify the difference between two elements. They should help to verify the identified performance properties that have to be verified, e.g. maximum/minimum values.
- *Intrinsic indicator*: since by definition such performances do not depend on the end-to-end temporal behaviour of the system, and are not a function of the state vector: they are very useful to verify internal behaviour of the system. The most usual intrinsic performance indicators for closed-loop control systems are: the stability and the stability margins, which require to be carefully defined according to the nature and the complexity of the system, the transient response properties such as overshoot and damping ratio. Other interesting performance independent from the outputs of a simulation stability are observability, controllability (defined in this paragraph), robustness, disturbance rejection can be computed out of the simulation and verified for analysis. Other properties find confirmation from the simulation: e.g., margins of stability or disturbances rejection.

In the next paragraphs the most important definitions for extrinsic and extrinsic performance are given.

2.6.1.1 Extrinsic performance parameters

The most common extrinsic performance indicators are:

- *Absolute Performance Error (APE)*: is the instantaneous values of the performance error (difference between desired-actual state) at any given time;

- Mean Performance Error (MPE) is the mean value of the performance error within a specified interval of time;
- Relative Performance Error (RPE) is the difference between the instantaneous performance error at a given time and its mean values over a interval of time containing that time;
- Performance Stability Error (PSE) is defined as the difference between the instantaneous performance error at a given time t and the error value at an earlier time $t-\delta t$, where δt is fixed;
- Absolute Knowledge Error (AKE) is defined as the instantaneous value of the knowledge error (defined as the actual value and the estimated/known value) at any given time;
- Mean Knowledge Error (MKE) is defined as the mean value of the knowledge error over a specified time interval;
- Relative Knowledge Error (RKE) is defined as the difference between the instantaneous knowledge error at a given time, and its mean value over a time interval containing that time.

2.6.1.2 Intrinsic performance parameters

Two main concepts are of interest within the discussion about the intrinsic performances of a controlled system: its stability and its robustness.

2.6.1.2.1 Stability

The “stability” is the intrinsic property defined as the ability of a system to remain indefinitely in a bounded domain around an equilibrium position or around an equilibrium trajectory when submitted to small external disturbances.

Considering a LTI system described in the Laplace domain by the equation $H(s) = C[sI - a]^{-1}B + D$, its characteristic polynomial is $p(s) = |sI - A| = \text{product}(s - \lambda_i)$ where λ_i are the eigenvalues of A or poles of $H(s)$. The system is

- Asymptotically stable if the real part of the eigenvalues are strictly minor than 0
- Marginally stable if there is at least an eigenvalue with real part equal to 0 and the others with real part strictly minor than 0
- Unstable if there is at least an eigenvalue with real part major than 0

Analogous definition can be made for LTI digital system, expressed in the z-domain. The system is

- Asymptotically stable if the real part of the eigenvalues is strictly minor than 1
- Marginally stable if there is at least an eigenvalue with real part equal to 1 and the others with real part strictly minor than 1
- Unstable if there is at least an eigenvalue with real part major than 1

There are popular criteria that allow determining the stability of a system both for the classic and modern control theory:

- The algebraic criteria: these criteria assume that the analytical expression of the characteristic polynomial of the system is available and give information with regard to the position of the roots of the characteristic polynomial in the left- or the right-half complex plane.[19]
- The Nyquist criterion: this criterion refers to the stability of the closed-loop systems and is based on the Nyquist diagram of the open-loop transfer function.[20]
- The Bode criterion: this criterion is essentially the Nyquist criterion extended to the Bode diagrams of the open-loop transfer function. [20]
- The Nichols criterion: this criterion, as in the case of the Bode criterion, is essentially an extension of the Nyquist criterion to the Nichols diagrams of the open-loop transfer function. [20]

- The root locus: this method consists of determining the root loci of the characteristic polynomial of the closed-loop system when one or more parameters of the system vary (usually these parameters are gain constants of the system).[21]
- The Lyapunov criterion: this criterion is based on the properties of Lyapunov functions of a system and may be applied to both linear and nonlinear systems. There are both a direct method and an indirect. [22].

Briefly, the algebraic criteria, the Nyquist criterion, the Bode criterion, and the Nichols criterion, as well as the root locus technique, are all criteria in the frequency domain. The Lyapunov criterion is in the time domain.

2.6.1.2.2 Robust stability

A control system is robustly stable if it is stable for every admissible perturbation: when stability is verified, it becomes of interest establishing the limit within this performance is guaranteed. It means to define the amplitude of uncertainties of the physical parameters describing the control system (plant, sensors, actuators, and controller) for which the closed loop remains stable. Pay attention that the stability margin shall also be defined for those controllers that are used during different phases of the mission for which the spacecraft characteristics or the objectives can present significant variations. In such cases the controller needs to operate properly over a certain range of plant behaviours rather than in disturbances condition: however the same analysis about stability margin can be led.

The robust stability analysis and the stability margins calculation generally pass through the computation of the so-called *sensitivity* and *complementary sensitivity* functions. From the Figure 29, it is easy to define:

- The Output Sensitivity as the closed-loop transfer function between the control reference r and the feedback error term ε
- The Input sensitivity as the closed-loop transfer function between the external disturbance d and the total action v .
- The Output complementary sensitivity as the closed-loop transfer function between the control reference r and the control performance y
- The Input complementary as the closed-loop transfer function between the external disturbance d and the control command u

The norm of these functions (in the frequency domain) helps to determine the stability margins. The norm definition is based the singular values σ . For a complex matrix A , the maximum singular value is given by

$$\sigma_{max}(A) = \max_{x \neq 0} \left(\frac{\|Ax\|_2}{\|x\|_2} \right) \quad [EQ.56]$$

The stability margins are determined by the maxima of these singular values over the frequency domain:

$$\begin{aligned} & \max_{\omega} \left[\sigma_{max} \left(T_{input}(\omega) \right) \right], \max_{\omega} \left[\sigma_{max} \left(T_{output}(\omega) \right) \right] \\ & \max_{\omega} \left[\sigma_{max} \left(S_{input}(\omega) \right) \right], \max_{\omega} \left[\sigma_{max} \left(S_{output}(\omega) \right) \right] \end{aligned} \quad [EQ.57]$$

The larger are these values, the smaller are the stability margins. As a consequence specifying a given level of stability margins can be achieved by specifying a maximum value for the singular values above.

Today, the controllers are implemented numerically, the controllers are not sensible to physical uncertainty and the margins requirement should be loosened – more or less – to account for that. However, whereas uncertainties are highly reduced at controller's level, they tend to increase at plant's level. In a GNC control loops, the dynamics of the satellites have grown in complexity over the past

years, due to large flexible appendages, large sloshing fuel masses and to more stringent or station-keeping pointing requirements. For these reasons, the stability margins are intended to cope with the growing uncertainties related to these elements.

In practice it can be difficult to ensure that this verification is fully exhaustive, due to the number of uncertain parameters and to the size of the domain that should be investigated. Scanning the full domain by a series of discrete sets of numerical values can lead to huge simulation times. Consider for illustration a simple (simplified) satellite dynamical model with a rigid central body and two steerable solar arrays with three flexible modes each: the elementary parameters required to describe this dynamics are

- the rigid inertia matrix of the full satellite (6 parameters),
- the cantilever frequencies for the flexible modes (6 parameters),
- the cantilever damping ratios (6 parameters),
- the modal coupling factors of the flexible modes (36 parameters, reducing to 12 considering pure modal shapes),
- the two steering angles.

Even considering a fixed, worst case damping ratio and pure modal shapes the sensitivity analysis should run over 26 elementary parameters, which makes it hardly manageable in practice. The search for a worst case of stability margins is partly driven by engineering feeling (for simple configurations and control laws, the smaller the inertia, the higher the coupling factor, the smaller the cantilever frequency often lead to minimum margins).

Systematic techniques exist based on advanced methods (for instance based on “M- \square decomposition” of the uncertain system), which allow – with some limitations – for a direct identification of the worst combination of uncertainties leading to the loss of the stability properties. Nevertheless these techniques are difficult to generalize and can reach their limits for systems with a large number of uncertain parameters; they cannot be set as a standard approach for verification.

2.6.1.2.3 Observability and controllability

The concepts of controllability and observability have been introduced by Kalman [24] and are of great theoretical and practical importance in modern control. For example, they play an important role in solving several control problems, such as optimal control, adaptive control, and pole placement.

Starting from [EQ.37] E [EQ38], the vector $x(t)$ is completely controllable or simply controllable if exists a piecewise (so without limitation on the amplitude or on the energy of $u(t)$) continuous control function $u(t)$ such as to drive $x(t)$ from its initial condition $x(t_0)$ to its final value $x(t_1)$ in a finite period of time.

The controllability is verified if the $rank(S)$ is maximum, where S is the controllability matrix $S = [B, AB, \dots, A^{n-1}B]$.

The output vector $y(t)$ is completely controllable or simply controllable if there exists a piecewise continuous control function $u(t)$, which will drive $y(t)$ from its initial condition $y(t_0)$ to its final value $y(t_f)$, in a finite period of time.

The controllability is verified if the $rank(S)$ is maximum; where S is the controllability matrix: $S = [D, CB, CAB, \dots, CA^{n-1}B]$.

The concept of observability is related to the state variables of the system and it is dual to the concept of controllability.

The state vector $x(t)$ is observable in the time interval $[t_0; t_f]$ if, knowing the input $u(t)$ and the output $y(t)$ for t contained in $[t_0; t_f]$, it is possible to determine the initial condition vector $x(t_0)$.

The observability is verified if the $rank(R)$ is maximum; where R^T is the controllability matrix: $R^T = [C^T, A^T C^T, \dots, (A^T)^{n-1} C^T]$.

2.6.1.2.4 Robustness [25]

Robustness is the property of a controlled system to achieve the control objectives against the disturbances and uncertainties. Two types of robustness can be considered: robust stability and robust performance.

As said before, a system is robustly stable if it is stable for each admissible perturbation. Moreover, a system performs robustly if it satisfies the performance specifications for all admissible perturbation. The stability and performance robustness depend on the controller, the adopted models and the set of perturbations.

Two main cases could be investigated to analyse the stability and performances robustness: the controlled systems with unstructured uncertainty and the controlled systems with structured uncertainty.

The unstructured uncertainties and perturbations can be categorized in Figure 36:

- Additive, that represents unknown dynamics operating in parallel w.r.t. to the system
- Multiplicative, that represents unknown dynamics operating in series with the system
- Feedback, that represents uncertainty on the closed loop control

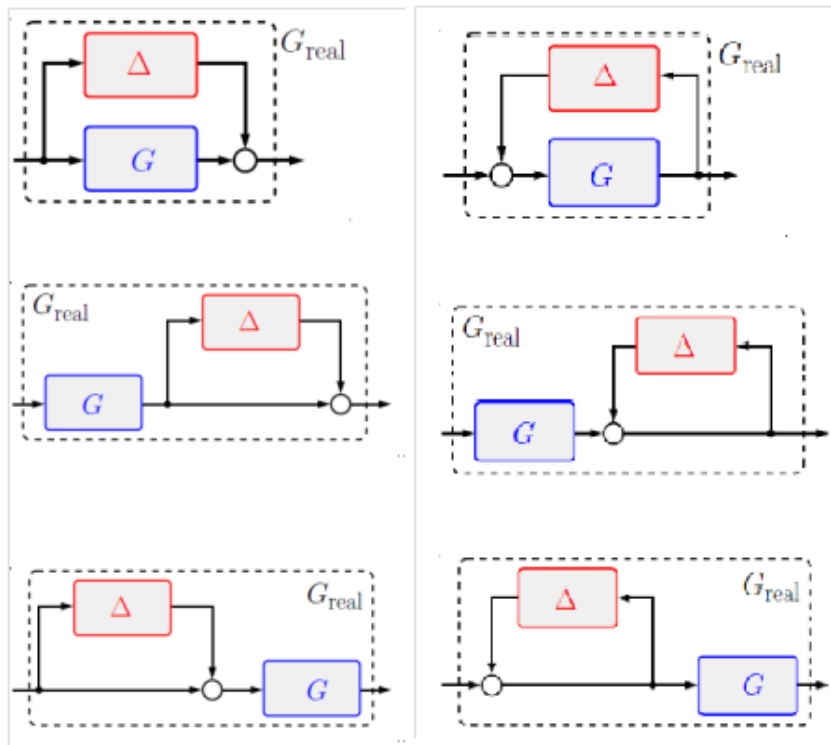


Figure 36: Additive and multiplicative unstructured uncertainties

Stability robustness and performance robustness can be evaluated if the perturbations are bounded $\bar{\sigma}[\Delta'(s)] \leq \Delta_{max}(s)$ where $\sigma(\cdot)$ is the maximum singular value and Δ' is any of the mentioned perturbations.

Taking into account the standard form of a general controlled system with uncertainty, a matrix description can be made:

$$\begin{bmatrix} Y_d \\ Y \\ M \end{bmatrix} = \begin{bmatrix} P_{ydw} & P_{ydw} & P_{ydu} \\ P_{ywd} & P_{yw} & P_{yu} \\ P_{mwd} & P_{mw} & P_{mu} \end{bmatrix} \begin{bmatrix} W_d \\ W \\ U \end{bmatrix} \quad [\text{EQ.58}]$$

The inputs are the perturbation input (W_d), the disturbance input (W), the control input (U); the outputs are the perturbation output (Y_d) the reference output (Y) and the measured output (M).

The perturbation bound is $\frac{1}{\Delta_{\max}(s)} = \bar{\sigma}[\Delta'(s)] \leq 1$ and normalized perturbation is defined as $\Delta(s) = \frac{1}{\Delta_{\max}(s)}\Delta'(s)$. The maximum singular value is $\bar{\sigma}[\Delta(s)] = \frac{1}{\Delta_{\max}(s)}\bar{\sigma}[\Delta'(s)] \leq 1$. The set of perturbation, for each frequency, that satisfies this bound is:

$$\|\Delta(s)\|_{\infty} \leq 1 \quad [\text{EQ.59}]$$

The normalized perturbation is incorporated into the system model by substituting $\Delta'(s) = \Delta_{\max}(s)\Delta(s)$. The stability robustness of controlled systems with uncertainty is addressed by the study of the standard model. Defining $N(s)$ as the nominal closed loop system, this equation is derived:

$$U(s) = K(s)M(s) = K(s)P_{mwd}(s)W_d(s) + K(s)P_{mw}(s)W(s) + K(s)P_{mu}(s)U(s) \quad [\text{EQ.59}]$$

From this equation, it is obtained:

$$\begin{bmatrix} Y_d \\ Y \end{bmatrix} = \begin{bmatrix} P_{ydw} + P_{ydu}[I + KP_{mu}]^{-1}KP_{mwd} & P_{ydw} + P_{ydu}[I + KP_{mu}]^{-1}KP_{mw} \\ P_{ywd} + P_{yu}[I + KP_{mu}]^{-1}KP_{mwd} & P_{yw} + P_{yu}[I + KP_{mu}]^{-1}KP_{mw} \end{bmatrix} \begin{bmatrix} W_d \\ W \end{bmatrix} = \begin{bmatrix} N_{ydw} & N_{ydm} \\ N_{ywd} & N_{yw} \end{bmatrix} \begin{bmatrix} W_d \\ W \end{bmatrix} \quad [\text{EQ.60}]$$

$N(s)$ is stable because the controller has been designed for the nominal control and the perturbation is stable because it has a bounded gain.

A general feedback system where perturbation is bounded ($\|\Delta(s)\|_{\infty} \leq 1$) is robustly stable for all possible perturbations because the nominal closed-loop system is stable and it is valid the theorem of “the small gain”:

$$\|N_{ydw}\|_{\infty} = \sup(\bar{\sigma}[N_{ydw}(s)]) < 1 \quad [\text{EQ.61}]$$

In many applications, additional constraints on the set of admissible perturbations are available. These constraints add “structure” by conferring a more general form of uncertainty than the unstructured. In fact, structured uncertainties arise when multiple perturbations affect the system.

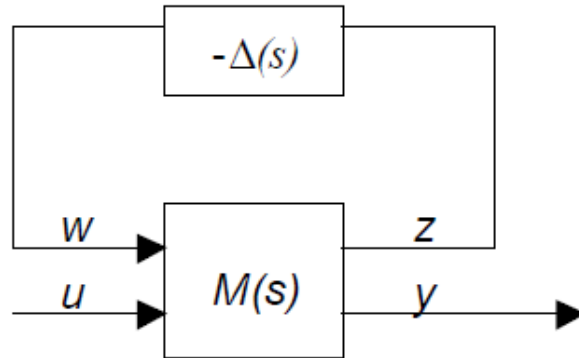


Figure 37: Structured uncertainties

From the mathematical point of view, the structured perturbation $\Delta(s)$ can be written as a diagonal transfer function matrix:

$$\Delta(s) = \begin{bmatrix} \Delta_1(s) & 0 & 0 & 0 \\ 0 & \Delta_2(s) & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \Delta_n(s) \end{bmatrix} \quad [\text{EQ.62}]$$

The structured perturbation is normalized so that its infinity norm is bounded by 1: $\|\Delta(s)\|_\infty \leq 1$ and all the blocks of the perturbation are scaled so that their infinity norm is bounded by 1:

$$\|\Delta_1(s)\|_\infty \leq 1; \|\Delta_2(s)\|_\infty \leq 1. \quad [\text{EQ.63}]$$

The stability of a system subject to a structured uncertainty is determined by analysing the feedback system in Figure 38. The nominal closed loop system is assumed to be stable. Any unstable poles of this system are therefore caused by closing the loop through the perturbation and are solution of $\det[I + N_{ydw d}(s)\Delta(s)] = 0$.

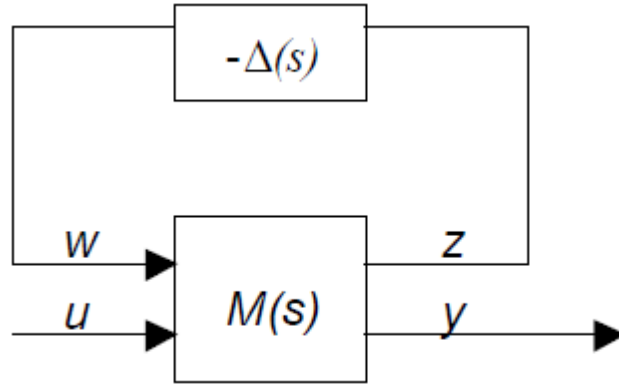


Figure 38: System with a structured uncertainty

In this case, stability robustness may be evaluated by the determination of the smallest perturbation that results in a pole with a non negative real part (for the continuous systems) or a real part higher than 1 (for discrete systems). Through mathematical calculations, the “size” of the smallest perturbation that destabilizes the system is defined as follows:

$$\inf\{\min_{\Delta(s) \in \bar{\Delta}}[\bar{\sigma}[\Delta(s)]] \text{ such that } \det[I + N_{ydw d}(s)\Delta(s)] = 0\} \quad [\text{EQ.63}]$$

The maximum Singular Value (SV) is a measure of the size of the perturbation.

A system is robustly stable if and only if smallest destabilizing perturbation is greater than 1. It means:

$$\inf\{\min_{\Delta(s) \in \bar{\Delta}}[\bar{\sigma}[\Delta(s)]] \text{ such that } \det[I + N_{ydw d}(s)\Delta(s)] = 0\} > 1 \quad [\text{EQ.64}]$$

However this problem is too complex, so the robust stability condition is put into a more useful form for both application and computation:

$$\sup\left\{\frac{1}{\min_{\Delta(s) \in \bar{\Delta}}[\bar{\sigma}[\Delta(s)]] \text{ such that } \det[I + N_{ydw d}(s)\Delta(s)] = 0}\right\} < 1 \quad [\text{EQ.65}]$$

This result is very similar to the form obtained for unstructured perturbation. The term within the bracket is called Structured Singular Value (SSV) and is formally defined as:

$$\mu(N) = \left\{ \frac{1}{\min_{\Delta(s) \in \bar{\Delta}}[\bar{\sigma}[\Delta(s)]] \text{ such that } \det[I + N_{ydw d}(s)\Delta(s)] = 0} \right\} \quad [\text{EQ.66}]$$

$$\mu(N) = 0 \text{ if } \det[I + N\Delta] \neq 0 \forall \Delta \in \bar{\Delta}$$

A general feedback system is robustly stable for all possible perturbations $\Delta(s) \in \bar{\Delta}$ and $\|\Delta(s)\|_\infty \leq 1$ if and only if the nominal closed loop system is stable and $\sup\{\mu[N_{ydw d}(s)]\} < 1$.

Performance robustness analysis can be based on the SSV as for the robust stability analysis. In fact, a particular method of specifying performance is to limit the ∞ -norm of the closed loop transfer function $\|H(s)\|_\infty \leq 1$ where $H(s)$ is the perturbed closed-loop transfer function here defined:

$$H(s) = N_{ywd}(s)[I - \Delta(s)N_{ydw}(s)]^{-1}\Delta(s)N_{ydw}(s) + N_{yw}(s) \quad [\text{EQ.67}]$$

The robust performances are reached if the system is robustly stable and $\|H(s)\|_\infty \leq 1$ is true for all admissible perturbations. The ∞ -norm cost function is typically used to specify performance robustness because it yields a robustness test that is easily applied in practice. The conditions for performance robustness can be precisely stated in terms of these transfer functions:

$$\sup\{\mu[N_{ydw}(s)]\} < 1 \text{ and } \|H(s)\|_\infty \leq 1 \quad [\text{EQ.68}]$$

This problem can be converted into the equivalent robust stability problem appending an uncertainty block to the system. The system in Figure 39 (a) meets the performance robustness objective if the system in Figure 39 (c) is robustly stable.

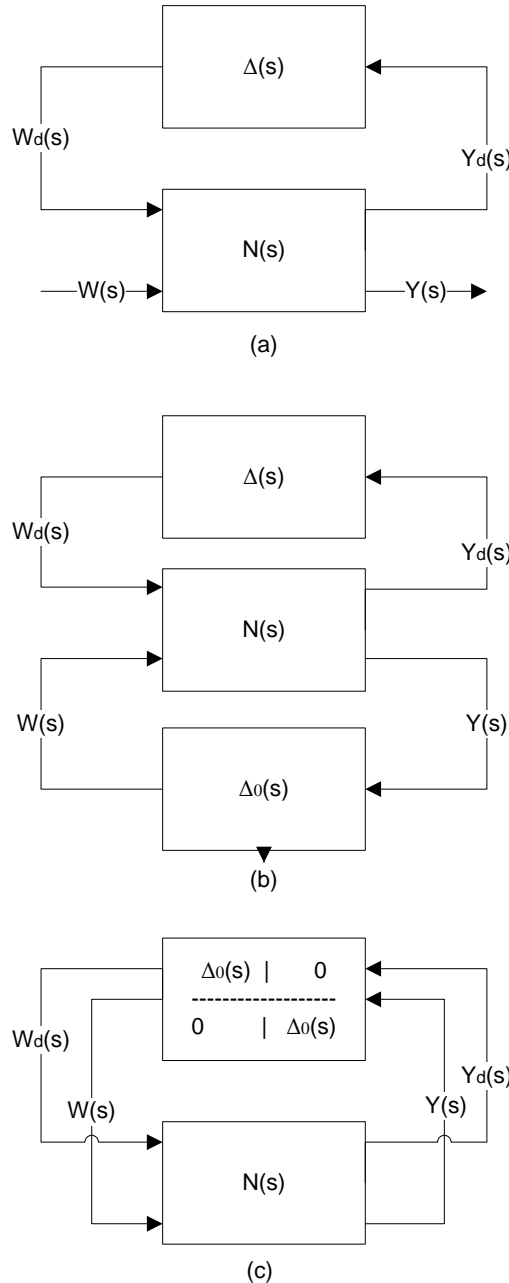


Figure 39: Performance robustness analysis using SSV

Robust stability of the system with the performance block implies that the system is stable for all the perturbation of the type:

$$\Delta_p(s) = \begin{bmatrix} \Delta(s) & 0 \\ 0 & \Delta_0(s) \end{bmatrix} \quad [\text{EQ.70}]$$

such that $\|\Delta(s)\|_\infty \leq 1$ where $\Delta_0(s)$ is a perturbation for which the robust stability is equivalent to the performance requirements. The system is robustly stable when subject to the perturbation if and only if $\sup\{\mu[N_{ydw}(s)]\} < 1$, which satisfies the first condition for robust performance.

The system with performance block is robustly stable proving that $\sup\{\mu[N(s)]\} < 1$. A consequence of the definition of the SSV is that $\sup\{\mu[N(s)]\} < 1$ is true if and only if $\det[I - N(s)\Delta_p(s)] > 0$ for all the frequencies and all admissible perturbations $\Delta_p(s) \in \bar{\Delta}_p$ such that $\|\Delta_p(s)\|_\infty \leq 1$.

Developing the determinant, it is possible to demonstrate that the performance robustness is satisfied if and only if (theorem of “the small gain”):

$$\left\| N_{yw}(s) + N_{ywd}(s)\Delta(s)[I - N_{ydw}(s)\Delta(s)]^{-1}N_{ydw}(s) \right\|_\infty < 1 \quad [\text{EQ.71}]$$

for all the unstructured perturbation $\Delta_0(s)$.

2.6.1.3 Practical aspects

In general, the complete sequence for stability, stability margins verification, robustness, and performance analysis for a real space system can be split into:

- linearization of the system (when possible) in the neighborhood of its operational conditions,
- design and tuning of the controller with respect to the linearized system,
- verification of the system stability properties (stability, margins, robustness, advanced methods) using the linear analysis techniques described in the previous clauses, taking into account the parametric uncertainties of the system
- final validation by performing time simulations with the complete system (including non-linear features), analyzing the response signal behavior.

Chapter 2 reference

1. J. Wertz, W.Larson, *Space Mission Analysis and Design*, 3rd Edition, Space Technology Library, ISBN 1-881883-10-8, 1999
2. Computer Science Corporation (Members of the Technical Staff Attitude Systems Operation), *Spacecraft Attitude Determination and Control*, J. R. Wertz, Ed. Dordrecht, The Netherlands: D. Reidel Publishing Company, 1978.
3. W.E.Hammond, *Space transportation: a systems approach to analysis and design*, Volume 1, AIAA Education series, 1999 ISBN 1-56347-472-7
4. ECSS-E-ST- 60-30C – *Space Engineering: Satellite attitude and orbit control system (AOCS) requirements*, (31/08/2013)
5. ECSS-E-ST-10C - *Space Engineering: System Engineering general requirements* – (06/03/2009)
6. ECSS-E-HB-60A - *Space Engineering Control engineering handbook* – (14/12/2010)
7. Bong Wie, *Space Vehicle Dynamics and Control*, Published by American Institute of Aeronautics and Astronautics, Inc., 2008, ISBN 978-1-56347-953-3
8. Fhese, *Automated Rendezvous and Docking of Spacecraft*, Cambridge University Press, 2003, ISBN 0 521 82492 3
9. Bak T., *Spacecraft Attitude Determination: A Magnetometer Approach*, Aalborg Universitetsforlag, 1999.
10. FL Markley, D Mortari, *Quaternion attitude estimation using vector observations*, Journal of the Astronautical Sciences 48 (2), 359-380, 2002
11. F. L. Markley, *Fast Quaternion Attitude Estimation from Two Vector Measurements*, Journal of Guidance, Control, and Dynamics, Vol. 25, No. 2, March–April 2002, pp. 411-414
12. M. Grewal, A. Andrews, *Kalman Filtering Theory and practice using Matlab*, Wiley Interscience publication, 2002

13. T. Humphreys, *Attitude determination for small satellites with modest pointing constraints*, 2002
14. M. D. Shuster and S. D. Oh, *Three-Axis Attitude Determination from Vector Observations*, *Journal of Guidance and Control*, Vol. 4, No. 1, January–February 1981, pp. 70–77.
15. C. D. Hall, *Book Review of Engineering Analysis in Applied Mechanics*, by J. W. Brewer, *Journal of Guidance, Control, and Dynamics*, 2003, Vol. 26, No. 4, 669-670
16. V. Chobotov, *Spacecraft Attitude Dynamics and Control*, Krieger Publishing Company, 1991, ISBN 0-89464-031-3
17. John F. Hanaway, Robert W. Moorehead, *Space Shuttle Avionics*, National Aeronautics and Space Administration Office of Management, 1988, Scientific and Technical Information Division
18. ECSS-E-HB-60-10A- *Space Engineering- Control performance guidelines* – 14/12/2010
19. M. Dekker, *Modern Control Engineering*, Marcel, Dekker, Inc., 270 Madison Avenue, New York, ISBN: 0-8247-8981-4, 2002, ISBN :0-8247-8981-4
20. R. Gessing, *Control fundamentals*; Silesian University of Technology; ISBN 83-7335-176-0, 2004
21. *The General Problem of the Stability of Motion*, Taylor & Francis, London 1992
22. E. Gilbert, *Controllability and Observability in in multivariable Control Systems*, J.S.I.A.M. CONTROL, Set. A, Vol. 2, No.1, Printed in U.S.A., 1963
23. J. Burl, *Linear optimal control, H₂ & H_∞ method*, Addison, Wesley, 2000, ISBN 0-20180-8684
24. L. B. Rainey, *Space Modeling and Simulation Roles and Applications Throughout the System Life Cycle*, The Aerospace Press El Segundo, California American Institute of Aeronautics and Astronautics, Inc. Reston, Virginia, 2004
25. R. Burns, *Advanced control Engineering*, printed by Butterworth Heinemann, 2001, ISBN 0750651008
26. J. Rohde, *Kalman filter for attitude determination of student satellite*, MD thesis Norwegian University of Science and Technology Trondheim, 2007
27. J. C. Doyle, K. Glover, P. Khargonekar, B. A. Francis. *State-Space Solutions to Standard H₂ and H_∞ Control Problems*. IEEE Trans. Automat. Contr. , vol. 34, Agosto 1989.
28. F. Donati, M. Vallauri . *Guaranteed Control of “Almost-Linear” Plants*. IEEE Trans. Automat. Contr. , vol. AC-29, no. I, pp. 34-41, Gennaio 1984.
29. M. Farza, M. M'Saad and L. Rossignol. *Observer Design for a Class of MIMO Non-Linear Systems*. Automatica, vol. 40, no. 1, pp. 135-143, Gennaio 2004.
30. K. Zhou. *Comparison between H₂ and H_∞ Controllers*. IEEE Trans. Automat. Contr., vol. 37, no. 8, Agosto 1982.
31. Wahba, G. Problem 65–1: A Least Squares Estimate of Spacecraft Attitude, *SIAM Review*, 1965, 7(3), 40
32. Davenport, P. B. "A Vector Approach to the Algebra of Rotations with Applications", NASA X-546-65-437, November 1965.
33. Keat, J. "Analysis of Least Squares Attitude Determination Routine DOAOP", CSC/TM-77/6034, Computer Sciences Corporation, Lanham-Seabrook, Maryland, February 1977.
34. Shuster, M. D., and Oh, S. D. "Three-Axis Attitude Determination from Vector Observations", *Journal of Guidance and Control*, Vol. 4, No. 1, 1981, pp. 70-77
35. Markley, F. L. "Attitude Determination Using Vector Observations: A Fast Optimal Matrix Algorithm", *Journal of the Astronautical Sciences*, Vol. 41, No. 2, April-June 1993, pp. 261-280.
36. Mortari, D. "ESOQ: A Closed-Form Solution to the Wahba Problem," *Journal of the Astronautical Sciences*, Vol. 45, No. 2, April-June, 1997, pp. 195-204.
37. Mortari D., "ESOQ-2 Single-Point Algorithm for Fast Optimal Spacecraft Attitude Determination", 7th Annual AAS/AIAA Space Flight Mechanics Meeting, Huntsville, AL, Feb. 10-12, 1997.

Chapter 3. The simulator

The simulator is part of a wider methodology that derives from the idea to improve the process of design, verification of the requirements and their validation against the environment of complex systems, with particular interest in the GNC subsystem of space vehicles. The improvement refers to the effectiveness and the low cost of the process that carries out to prevent misbehaviors because allow individuating errors in the specification, the design, the manufacturing, and the integration phases. The methodology puts together the modern approach of the SE, forsaking the traditional document-centric approach, and the M&S based approach. The methodology enables the design and the V&V of a system through an iterative process that leads to verify the entire system against its specifications as soon as possible, without the necessity to have the availability the real software and real hardware. To help this process, all the activities have to be made on a common “platform” in any moment for mitigate the risk of incompatibility between successive stages as well as misunderstandings and loss of information. In order to avoid these problems, the methodology leans on a simulation tool called Simulator (in particular, StarSim) able to support the decision making activities and the verification of functional and operational requirements. In fact, the simulator guarantees the possibility to simulate the behavior of the system thanks to the constant availability of virtual models. This allows progressively introducing pieces of the on board software and hardware and maintaining as virtual the models of the not available equipment and not reproducible environmental phenomena. Moreover, the time sequence of verification algorithms, then software, and finally hardware could not be rigorously followed thanks to the use of hybrid simulation configurations. It means that hardware parts of the system can be inserted in the simulation loop leaving the other incomplete parts as virtual models managed by the simulator. Clearly, the simulator must have virtual models with different degrees of fidelity, suitable with the ongoing stage of the project as well as the capability to host and manage simulations with real software (intended as executable files properly generated) and/or real hardware (intended mainly as embedded systems, sensors, and actuators) in the loop. Before to start any session, the simulation architectures and configurations can be chosen like a customer can choose a product on the shelves of the supermarket. Following different strategies, the user (i.e. designer, developer, analyst, customer, or operator) puts into the “basket” what it is useful (models and interfaces) and then builds the simulation “loop” through the setup of the parameters of the selected models. From the user choices and setup, the simulator is able to generate a software code flexible enough and build a modular simulation structure in which every piece (virtual or physical model) can be replaced by a similar element without excessive effort. Moreover, the continuous use of the simulator allows to have a great number of data for analyses and comparisons and the capabilities of different configurations and setups allows to deeply verify the system for various conditions, increasing the confidence about the designed system.

3.1 The methodology

The Figure 40 shows the flow chart of the methodology. Going into details, three main steps characterize the methodology.

1. **design step** foresees the application of SE methods for the GNC design: mission, program and system requirements and constraints analysis leads to individuate the critical aspects that impact the GNC design. Moreover, the interaction between the GNC and the other subsystems shall be taken into account during the design process because the required performances (e.g. pointing accuracy), the power and mass budgets, the instrumentation and payload layout affect the GNC design. The main and addition functions of the GNC subsystem shall be individuated and analyzed: using SE engineering techniques like N2 diagram, product trees, or functions vs

equipment matrices the GNC functional and physical architectures are built, sensors, actuator and electronics devices are individuated. The GNC subsystem architecture is characterized by the control architecture from which derives the choice of guidance, navigation and control strategies (e.g. deterministic or recursive methods for attitude or trajectory determination, nominal and off normal control modes and the associated guidance strategies and control laws).

2. **M&S step** is the core of the methodology. It foresees the following steps:
 - a. *“Definition of environmental parameters, disturbances and noises characterization and S/C motion features deriving from mission analysis”*. Orbit parameters and orbit propagation, disturbance torques and forces, noises generated by the radiation and heat fluxes shall be known or estimated through virtual, hybrid or physical models according to the phase of the product life cycle.
 - b. *“GNC models choice and characterization”*: sensors, actuators, and electronic devices that constitute the GNC subsystem, derive directly from the system and control architecture. Their models shall be chosen accordingly: virtual models have different levels of accuracy and detail (simpler for the design and more complex for the verification) and the hybrid and physical models (the real objects) can be MU, EM, QM, and FM. The chosen models shall be properly assembled following the architecture and maintaining the loop/chain sequence.
 - c. *“Interface choice and characterization”*. Particular attention shall be paid for the interfaces among the models or elements. A certain numbers of software and hardware interfaces shall be available in order to guarantee the connection for virtual, hybrid and physical models. These interfaces shall be properly selected reflecting the design.
 - d. *“GSE choice and characterization”*. GSE involved in the simulation session for stimulation, for power supplying and/or for connecting hybrid and physical models shall be chosen. For SIL, CIL and HIL simulation, they are essential.
 - e. *“Special functions and GNC strategies choice and/or implementation”*. They can be taken from database or defined accordingly the GNC design. “Special functions” refers for example to mathematical operation (i.e. rotation matrix, eigenvalues computation, 3x3 cross-product, etc...); GNC strategies are the algorithms that implement attitude and orbit determination and control, and control modes. Examples are q-est, ESOQ, q-method algorithms and Kalman Filters (for navigation/determination), detumbling mode, pointing mode, chasing mode (for guidance), PID, LQR, Fuzzy logic control (for control techniques).
 - f. *“Setup of models, interfaces, and GSE”*. This operation shall be made before the simulation start and requires the definition of parameters:
 - i. for equipment, GSE models and special functions, it means tuning parameters according to the data sheets, system/mission specifications or GNC design. For hybrid and physical models, the set up activities foresees the plug in of cables and connectors and their proper arrangement within the simulation facility room.
 - ii. for interface: it means to select the protocols, data flow control, speed, the generation of interrupt and so on.
 - g. *“Configuration of simulation parameters”*. Real time (RT) vs not real time (not RT) execution choice, numerical setting (e.g. the integrator type and integration method), frequency of the simulation loop, simulation duration, initial condition of the each variable or constant value out of the models, interfaces, GSE, special functions settings are some examples.

- h. *“Selection of data outputs”*. Data outputs are all those parameters that will result of interest for real time and post-processing analysis. More in general, all the data repository tags and memory locations shall be defined.
 - i. *“Simulation execution and the live data visualization”*. At the end of every simulation loop, the outputs and the simulation parameters can be updated, stored and visualized in order to guarantee the monitoring and control from the user.
- 3. Verification & validation step** is performed both during and after the simulation session execution through the analysis and the comparison of the outputs, that lead to the evaluation of the GNC functions, performances and strategies (determination and control). Failures, errors, bugs, wrong implementation reports as well as the requirements satisfaction are possible outputs of this step. In the latter case the methodology process is completed and no new iterations should be scheduled. On the contrary, if the requirements are partially or not satisfied a new iteration must be performed, changing some part of the design. The outputs can be also used in support to the requirements negotiation with the program manager or system engineer: in this case the system (and mission, if any) requirements analysis shall be reviewed. If the negotiation is not possible the methodology restarts from review of the subsystem design, modifying parameters, strategies, hardware and/or software parts and modelling and simulating again the behavior of the new solution in order to have new data and evaluate again the requirements satisfaction.

The methodology is characterized by focal concepts, listed hereafter:

- *“The speed to complete an iteration”*. Since for every design “attempt” a model is ready, a simulation session can be performed and the results are soon available either for verification of functions and operations or to take a decision.
- *“Unique platform”*: all the methodology steps are led within the same platform in any phase (forever and ever!). In this way, the procedure to follow is always the same for any type of action:
 - to introduce parameters
 - to select and/or build models
 - to setup the simulation
 - to execute the simulation
 - to gather & compare results

All these activities require the knowledge of a limited numbers of actions/operations and more simplicity to train users.

- *“The possibility to anticipate software bugs and hardware defects”*. OBSW can be developed and tested through the simulator before the loading on the on board computer: it allows fixing bugs on the coding as well as the implementation of functions and algorithms. The capacities of an hardware circuit can be tested before it is manufacturing through virtual models and/or emulators.
- *“The reduction of the time to market”*: it means reduce the length of time it takes from a product being conceived until its being available for sale or for accomplish its “mission”. This is because the possibility to reproduce the behavior of an element allows to “test” it also when it is not available or during its acquisition time.

To conclude, the methodology aims at improve the effectiveness of the long and essential process of design and operational and functional verification of a complex system, avoiding:

- *Losses of time*: having a unique platform saves the time spent in boring activities as translate algorithms from a code to another; sharing directly models reduces the time to tedious and humbling activities as read and write documents.
- *Losses of money*: reducing the production of prototypes substituted by virtual models, and the necessity to physically reproduce phenomena to effectively validate the system.
- *Waste of human resources*

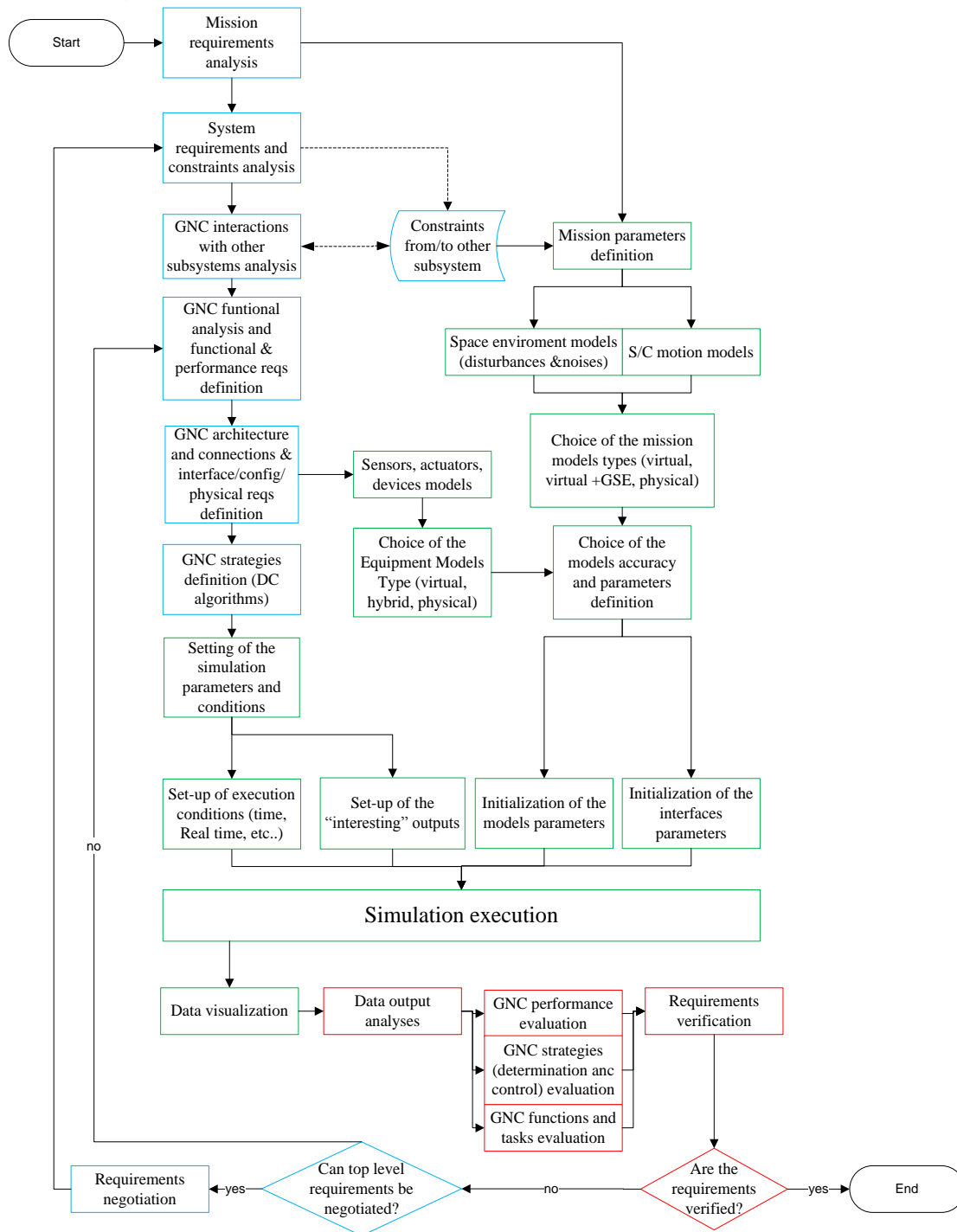


Figure 40: Flow chart of the methodology

3.2 StarSim simulator

The simulator StarSim is the key element of the methodology, it has been designed and built to satisfy the top level requirements, remaining within the programmatic and technical constraints (see 3.2.3) for all the StarSim requirements release v.1. The design drivers and the main expected features have been investigated, individuated, studied, and analyzed in order to produce a simulator that meets the requests of the methodology.

Key features and drivers for the simulator design are listed hereafter [1], [2], [3].

- **High speed and Real Time execution** means that the simulator shall work both not-real time simulations, that require high execution speed, and real time simulations, that require executing the processment or the handling with an established precision (hard real time vs soft real time). Not real time simulations are performed especially in the first phases of the project (feasibility and preliminary design) when quick solutions are requested and during the verification of algorithms. In both cases simulation sessions shall be as quick as possible to perform and compare alternative solutions. On the contrary, real time execution is needed when real software and real hardware are involved in the loop. In this case the simulation shall hold the time imposed by hardware and software: it shall provide information or stimuli not only correct from a logical point of view but also on the “instant” (with a tolerance) in which they are expected.
- **High flexibility** is a feature required in the simulator devoted to the V&V of complex systems which are constituted by a lot of parts or elements that could/should be changed/updated to verify different configurations. Moreover, the same element can be represented in different way in relation to the current status of the project: it could be a simple mathematical model, a complex virtual model, a piece of software code, a physical hardware or the final, entire system. These issues require a “modular” architecture where each part is an “object” or “block” that can vary and can be interconnected with other elements in different ways. The capability to manage a great number of configurations, varying parts of software or hardware elements and interfaces, bestows modularity on the simulator.
- **High connectivity** means that all the elements can be connected through logical and physical interfaces on which the information pass formatted according standard or custom protocols. Standard protocols are generally preferred for low level communication: serial, USB, TCP/IP are example of very common interface already addressed in COTS microprocessors embedded boards, PCs and networks of PCs. Custom protocols are preferred to format data, especially when coding and encrypting operations are needed. In any case, simulator can connect elements starting from database of standard interfaces and giving the possibility to define custom protocols.
- **High fidelity wrt real world:** the simulator shall perform simulations with a high degree of fidelity. The fidelity defines how accurately a model represents the behavior of an element (e.g. equipment or environment) it is modeling. Different level of accuracy can be defined:
 - *Accurate:* concepts that are modeled to a declared tolerance. Such tolerances should be stated explicitly. The normal values for telemetry parameters dependent upon this model should be within limits (if defined).
 - *Emulated:* Simulating specifically processors allow the real software code/image to run inside the simulation.
 - *Exact :* Used to describe concepts for which a zero tolerance is applicable. This is normally applicable to discrete systems.
 - *Functionally:* Functionally modeled units/functions should work/ behave as the real unit/function with respect to their external interfaces.

- *Plausible or Realistic*: Variables that should be modeled such that trends can be observed in their behavior in relation to outside influence without being precisely modeled to a declared tolerance.
- *Representative*: Data described as representative does not need to be modeled; pre-set value should be provided within the measurement range of the parameter. This value will always be used by the simulator unless updated from the simulator console, when desired.
- *Static*: Fixed values only.

Out of the definitions, a high fidelity degree gives a high confidence in the results of the simulation wrt the real on orbit behavior. However, high fidelity is not the same of high detail. In fact, the high level of detail is not always necessary and, sometimes, it is useless or dangerous: high detail level carries out high complexity of the model. It causes a high request of computational resources, reducing the simulator performances and, in some case, compromises the correctness of the simulation session results. The simulator shall provide a various choice in dynamics database of models but the main effort is for the user that must think “what subset of reality does the simulator try to represent?”

- **Ease to manage simulation sessions**: the simulator shall manage the simulations from different architecture definitions, initialization of the parameters, and the results analysis and the verification of the requirements after the execution. All these operations shall be performed or autonomously or by the users that should be helped and supported in their activities (e.g. utilizing a friendly and intuitive GUI). Data shall be saved and stored to favor evaluations of performances or good working and comparisons between results of other sessions.
- **Effectiveness of the data**. It is fundamental to identify the input data (scenarios, boundary conditions, simplifying assumptions and criticalities) for the simulator and for each simulation session. Inputs enter in the simulator determining and influencing the outputs. Taking into account the proposition: “garbage in garbage out”, what inputs choose, what combination of data come up with the specification, what equations, parameters, and algorithms use to determine outputs from input are critical points on which pay attention.

3.2.1 Why build a *in-house* simulator?

One of the key for the verification of complex system is the choice among: the reuse of an existing simulator, the development of a simulator based on commercial software or the development of a new, own, customized simulator. The better solution is, in general, the reuse of a simulator previously developed and, possibly, already validated because it confers a higher level of confidence, and it permits to reduce the efforts of development and validation for the engineers and developers as well as the time of learning and training for the operators, analysts and users.

However, if the simulator does not exist what is the best solution? Using commercial software and organize them or designing and building a self developed simulator? Probably it is impossible to give a final answer. In the next lines, the choice of develop StarSim as a custom simulator is discussed.

The analysis started from the main research goals:

- To have a unique simulation platform able to support the M&S activities in every phase of the product life.
- To reduce the waste of resources and have self contained tools exclusively devoted and optimized for space missions.
- To maintain a low cost approach.

Commercial software costs are related to the acquisition of licenses. In many cases Software House present different types of license: the so called educational versions have lower cost or, even, they are free but educational versions shall not be used for industrial, commercial or military designs. On the contrary, the complete versions are absolutely useable in any kind of project but their costs are higher. In-house software for simulator can generally be based on free software platforms and programming languages (i.e. C/C++ or Java, and Linux as OS); their costs are mainly due to the development and consequently related to the human resources needed. However the human resources costs are present also considering commercial tools that require time-consuming training and learning activities.

Commercial software are as much as possible general purpose: they are overmuch, providing tools and models unused in specific field (i.e. managerial libraries are completely useless for a space vehicles simulator) but within the license the customer pays necessarily also for that. In other case, interesting tools or libraries are out of the basic licenses and they should be expressly bought with further costs. On the contrary, commercial general purpose software does not have sufficiently devoted tools for specific applications: to contain this disadvantage they provide development environment but this means programming efforts also with commercial software. Moreover, often commercial-off-the-shelf (COTS) products allow describing problems in a general way using objects and state variables. In many cases they are tailored to applications such as queuing theory, industrial engineering, or control systems; for these reason they are usually very effective for “quick and dirty” simulation studies but break down when a simulation problem becomes detailed enough to demand a lot of customizing.

Commercial tools do not generally cover all the phases in the product life cycle. In fact, well defined and effective tools exist for feasibility analysis or design or verification of algorithms, software, hardware but it is infrequent that a tool performs all these tasks together.

Other two crucial points remain opened: 1) how to share information and data between tools and development environments deriving by different providers, 2) how to guarantee the compatibility of tools among themselves and with the hardware architecture (e.g. processor), kernel, firmware and operating systems.

Remaining on the commercial tools, the idea can be put together different tools in the same platform: it surely can result in a monstrous effort!

A very recent solution is provided by the ensemble of UML and SysML. However, it seems that high quality solutions derive from the developments of a custom simulator with free and famous software, e.g. TASTE.

In-house simulators are developed independently and from a lower level so that they are defter and any modification and change results easier to be implemented because a higher control due to the direct development of the code is possible. Commercial software does not allow a complete management at the machine level even if they relieve the developer to enter in the complex management of the processes at processor level.

An important feature is the generation of the code for embedded applications: some tool allows developing, producing, compiling and linking the operative software that will be loaded on the embedded board or microprocessor in the final application. The most common solution is the “rapid prototyping”: it is ‘the process of quickly building and evaluating a series of prototypes [3]. The rapid prototyping requires less effort in the production of the code because it is automatically produced by the tool but rarely is directly ready for the compiling and clearly not even for the cross-compiling. Moreover, every modification always requires the restarting of the entire process and the generated code is not user friendly and it is less recommended that someone makes change on the rapid prototyping output. On the contrary, a well structured in-house simulator, as StarSim, favor the rapid prototyping for the software (also for the on board software) deriving from the user setup and permits to modify the generated code (obviously according the coherent rules).

In-house simulator allows to group all the features required along the entire design cycle within a unique platform, independent from the type of software and hardware that will be designed, developed and verified using the same simulator. Moreover, the simulator shall remain independent from the commercial tool, providing a self-contained solution for the design of a space product (in particular for GNC).

In-house simulator does not exclude the possibility to interface with commercial software: it would help the use of the in-house simulation within an industrial field, where in some case the internal tradition and background tends to be maintained: the in-house simulator could accept inputs generated from other software in previous projects and adapt them to execute simulations.

What about the validation and accreditation? Simulator shall be validated from the hardware, software and models point of view. This is a big issue that, in any case, also commercial tools cannot solve easily, because for example putting together two certified software does not automatically lead to a certified software. On the contrary, validating a unique platform is a big issue but once the capabilities are validated and accredited the simulator has a “secure core” from which updating database and solution and through which verified future products and design with a better confidence level.

3.2.2 The simulator functions

This paragraph contains the analysis of the functions that the simulator shall accomplish. The Functional Tree Analysis is the method used to perform the analysis.

Figure 41 shows the top level functions deriving from the methodology objective: “to facilitate the design and verification of algorithms, software and hardware” that constitute the system. How to realize that? The answer foresees the use of AIL,SIL,CIL,HIL simulations session along the product life cycle, the sharing of data repository, models, simulation sessions results among the team members, and the capability to get automatic some part of the development and verification (e.g. the code generations or the check if the requirements).

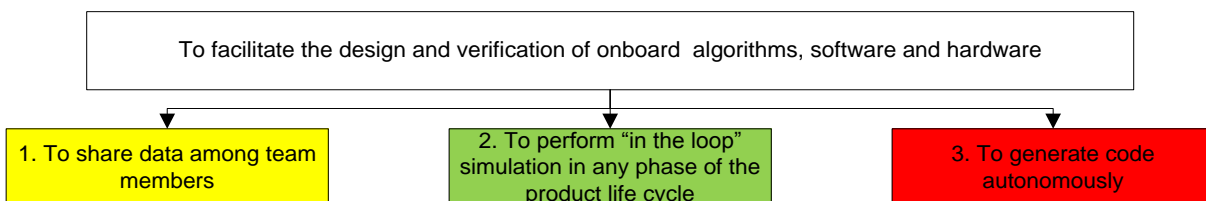


Figure 41: Top level simulator functions

Next figures going into details of the sub-functions derived from the top.

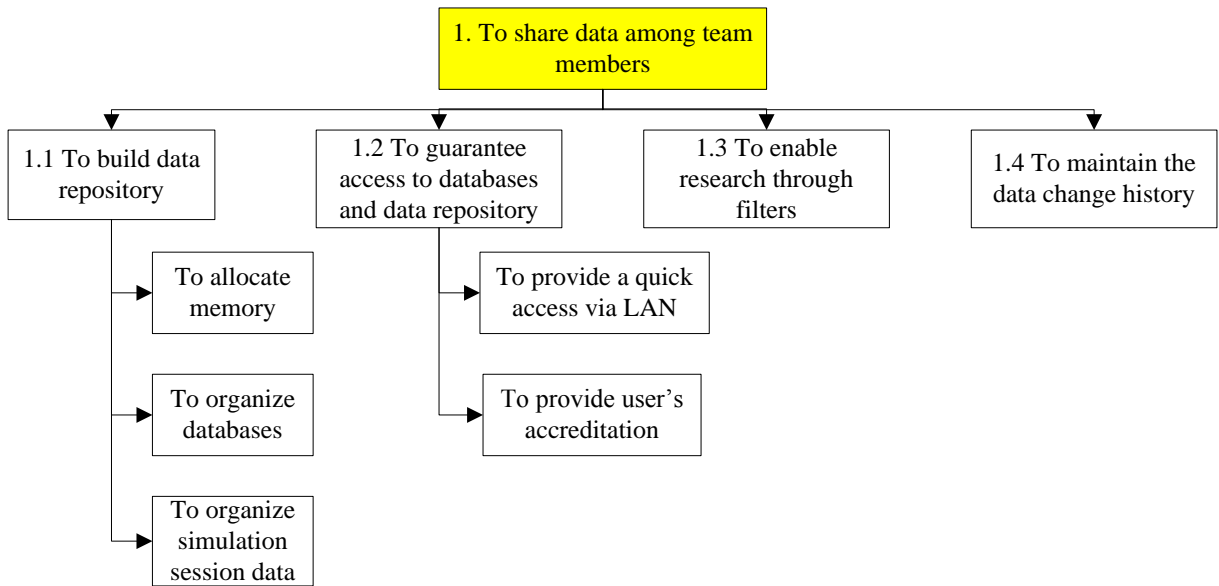


Figure 42: FT – subfunctions (I)

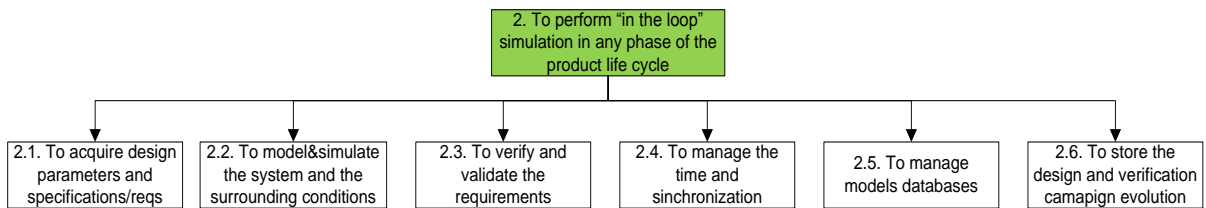


Figure 43: FT – subfunctions (II)

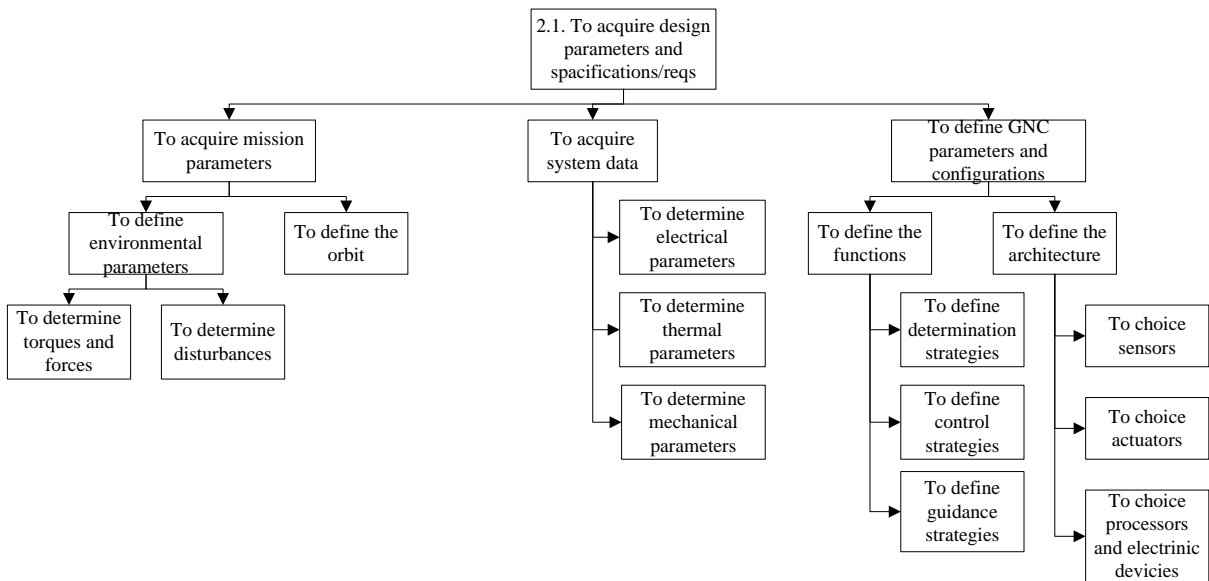


Figure 44: FT – subfunctions (VI)

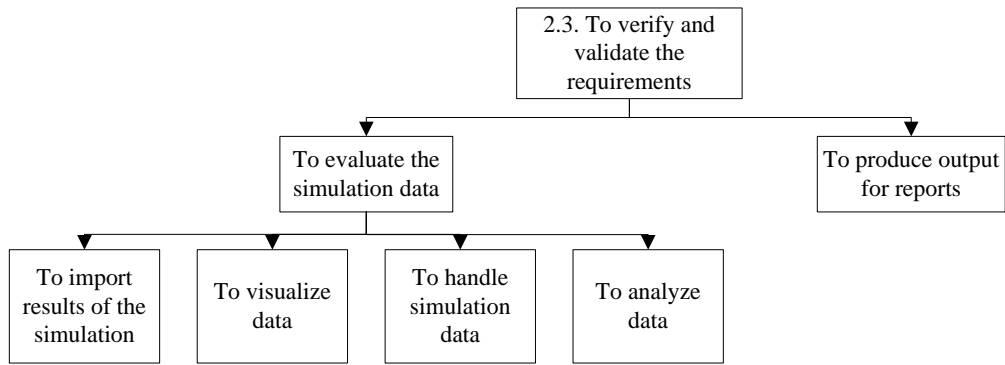


Figure 45: FT – subfunctions (III)

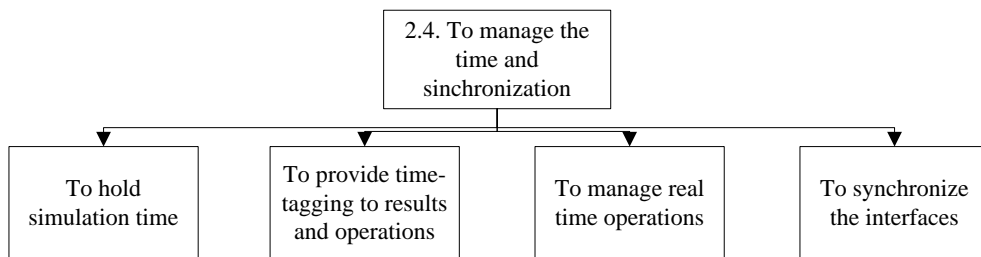


Figure 46: FT – subfunctions (IV)

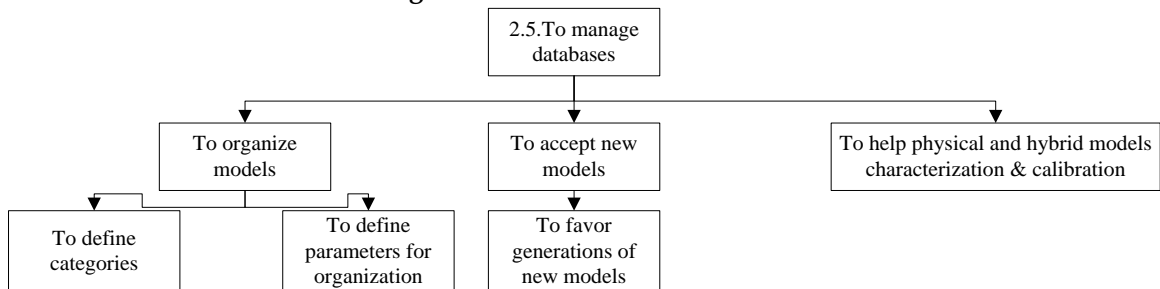


Figure 47: FT – subfunctions (V)

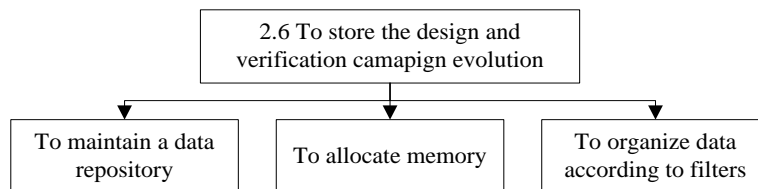


Figure 48: FT – subfunctions (VII)

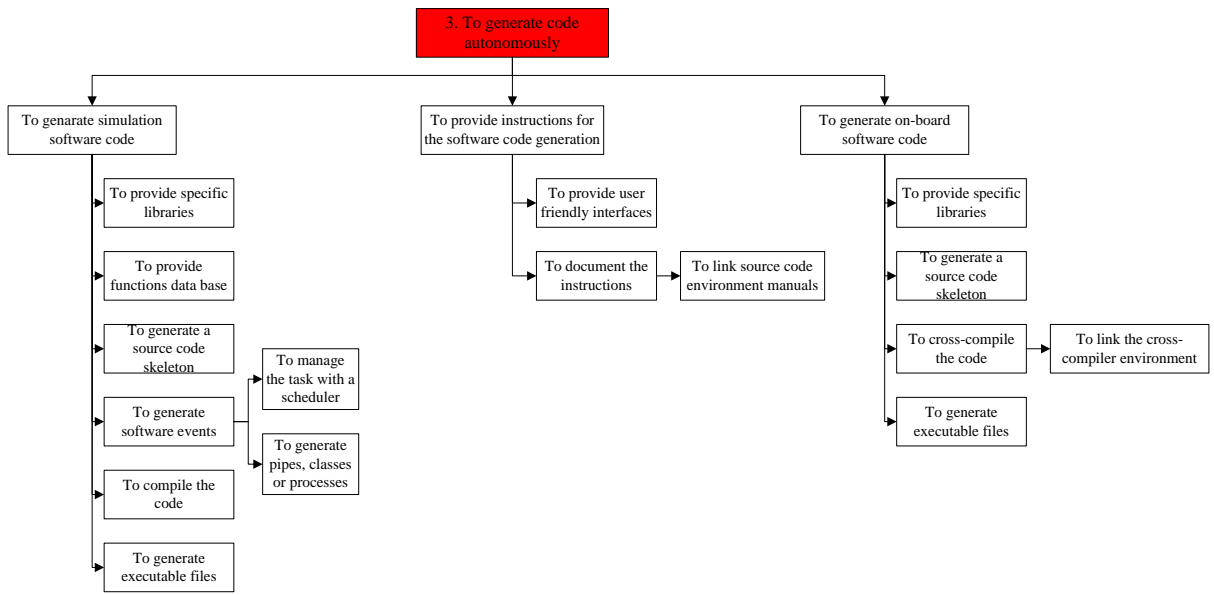


Figure 49: FT – subfunctions (VIII)

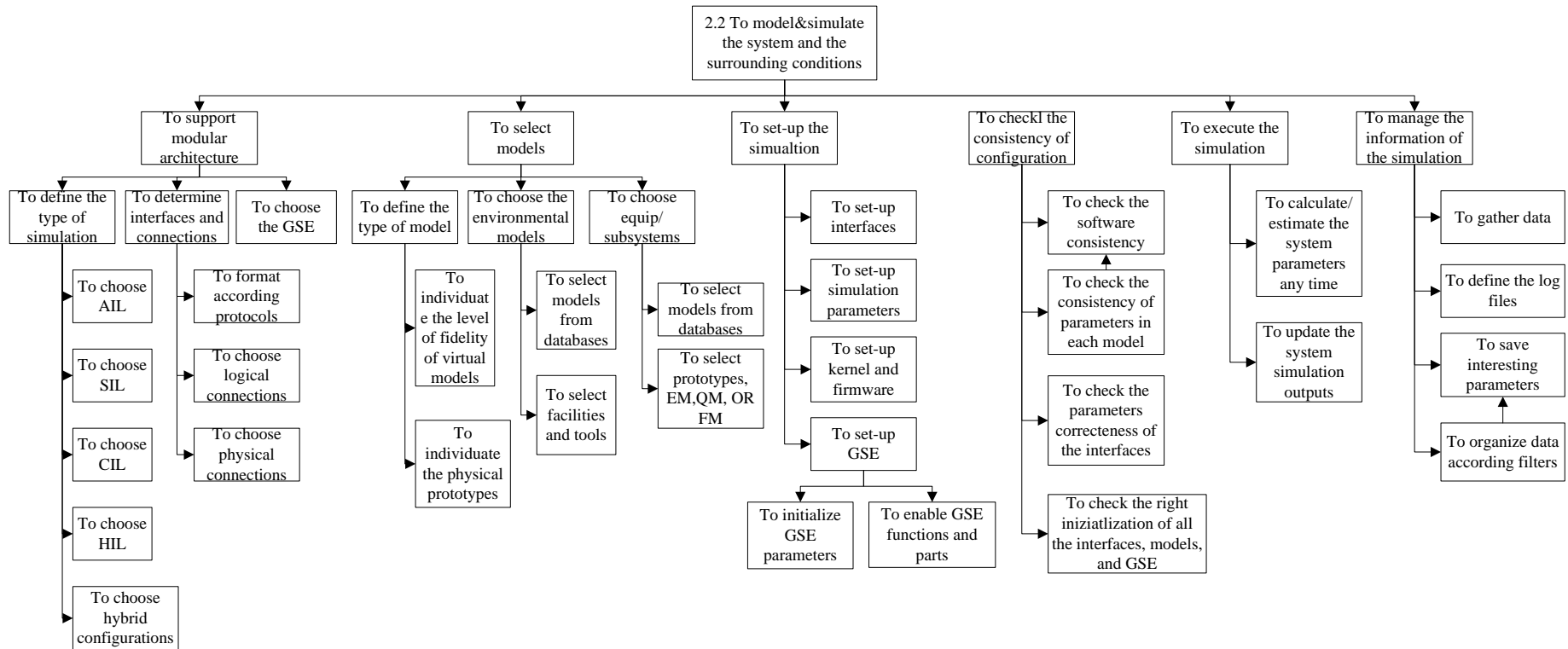


Figure 50: FT – subfunctions (IX)

3.2.3 Simulator requirements

ID	Proposition	from	to
<i>Top Level Requirements</i>			
STARSIM.TL.010	StarSim shall support the design and verification of complex system, according to the methodology		STARSIM.FN.0000
STARSIM.TL.020	StarSim shall be applied to a space system project frame		
STARSIM.TL.030	StarSim shall constitute a unique platform that must be used along the life cycle of the product developed with the methodology.		STARSIM.TL.031, STARSIM.TL.032, STARSIM.TL.033, STARSIM.TL.034
STARSIM.TL.031	StarSim shall be used for the feasibility study of space system	STARSIM.TL.030	
STARSIM.TL.032	StarSim shall be used for the preliminary and detailed design of space system	STARSIM.TL.030	
STARSIM.TL.033	StarSim shall be used for the production and qualification of a space system	STARSIM.TL.030	
STARSIM.TL.034	StarSim shall be used for the support to the on orbit operations of space system	STARSIM.TL.030	
STARSIM.TL.040	StarSim shall be independent by other tools or commercial software		STARSIM.TL.041
STARSIM.TL.041	StarSim shall be a self-contained platform. No extern and commercial tools, facilities and software shall be required for the StarSim good functioning	STARSIM.TL.040	
STARSIM.TL.050	StarSim v1.0 shall be configured for functional and operational requirements verifications of a space project, in particular for GNC subsystems		
STARSIM.TL.070	StarSim can interact, if required, with commercial software and tools according specific protocols and data repository rules		
STARSIM.TL.080	StarSim design, development, manufactuting shall be complinant with the ECSS, as much as possible		
STARSIM.TL.090	StarSim shall allow the reconfiguration of the simulation architecture		
STARSIM.TL.100	StarSim shall perform simulation in Real Time and Not-Real Time		

STARSIM.TL.110	StarSim shall guarantee that more simulation sessions can be run simulataneously		
STARSIM.TL.120	StarSim shall be flexible		STARSIM.CNF.800, STARSIM.TL.121
STARSIM.TL.121	StarSim databases and configurations should be updated by the user	STARSIM.TL.120	
STARSIM.TL.130	StarSim shall be a tool for developers, designers, analysts, users		
<i>Programmatic requirements</i>			
STARSIM.PR.010	ASSET of Politecnico di Torino and AVS shall take benefit from the methodology and the StarSim		
STARSIM.PR.020	The first version of the StarSim shall at least support the GNC project for different type of space missions and space vehicle, individuated as case studies.		
STARSIM.PR.030	StarSim v1.0 shall be designed, built and validated within the period January 2011-December 2013		
STARSIM.PR.040	StarSim v1.0 shall consider only low cost solutions		STARSIM.PR.041
STARSIM.PR.041	The StarSim v1.0 shall cost less than 10.000 euros (TBC) considering the equipment already available in the laboratories and facilities of the STARLAB at Politecnico di Torino and AVS. The costs of the personal shall not be taken into account	STARSIM.PR.040	
STARSIM.PR.050	StarSim v1.0 shall be delivered for the StarLab and mainly applied to small satellite missions		
<i>Functional requirements</i>			
STARSIM.FN.0000	StarSim shall facilitate the design and verification of on board algorithms, software and hardware	STARSIM.TL.010	STARSIM.FN.1000, STARSIM.FN.2000, STARSIM.FN.3000
STARSIM.FN.1000	StarSim shall share data among team members	STARSIM.FN.0000, STARSIM.TL.130, STARSIM.PR.010	STARSIM.FN.1100, STARSIM.FN.1200, STARSIM.FN.1300, STARSIM.FN.1400

STARSIM.FN.1100	StarSim shall manage data repository	STARSIM.FN.1000	STARSIM.FN.1110, STARSIM.FN.1120, STARSIM.FN.1130
STARSIM.FN.1110	StarSim shall allocate memory	STARSIM.FN.1100	
STARSIM.FN.1120	StarSim shall organize models database	STARSIM.FN.1100	
STARSIM.FN.1130	StarSim shall organize interfaces database	STARSIM.FN.1100	
STARSIM.FN.1200	StarSim shall guarantee access to database and data repository	STARSIM.FN.1000	STARSIM.FN.1210, STARSIM.FN.1220
STARSIM.FN.1210	StarSim shall provide quick access via LAN or from the command console	STARSIM.FN.1200	
STARSIM.FN.1220	StarSim shall provide the accreditation of the user	STARSIM.FN.1200	
STARSIM.FN.1300	StarSim shall enable researches of information according filters (data, type, value, etc...)	STARSIM.FN.1000	
STARSIM.FN.1400	StarSim shall maintain the data history of a project	STARSIM.FN.1000	STARSIM.FN.1410, STARSIM.FN.1420, STARSIM.FN.1430
STARSIM.FN.1410	Data of the same project shall be contained in folders of the StarSim called "name_program"\ "namedatasession"	STARSIM.FN.1400	
STARSIM.FN.1420	The configuration for each session is saved "name_program"\ "namedata" _CNF.txt	STARSIM.FN.1400	
STARSIM.FN.1430	The models flow for each session is saved "name_program"\ "namedata" _MDL.txt	STARSIM.FN.1400	
STARSIM.FN.2000	StarSim shall perform "in the loop" simulation in any phase of the product life cycle		STARSIM.FN.0000
STARSIM.FN.2100	StarSim shall acquire design parameters and specifications/requirements	STARSIM.FN.2000	STARSIM.FN.2110,
STARSIM.FN.2110	StarSim shall acquire the parameters characterizing the mission and the involved system(s)	STARSIM.FN.2100	STARSIM.FN.2111, STARSIM.FN.2112
STARSIM.FN.2111	Environmental & spacecraft motion parameters shall be defined	STARSIM.FN.2110	STARSIM.FN.2111.1, STARSIM.FN.2111.2
STARSIM.FN.2111.1	The torques and forces shall be sized	STARSIM.FN.2111	

STARSIM.FN.2111.2	Disturbances and noises shall be estimated for peaks, mean and trend	STARSIM.FN.2111	
STARSIM.FN.2112	The orbit parameters shall be determined	STARSIM.FN.2110	
STARSIM.FN.2120	StarSim shall acquire system data deriving from the design	STARSIM.FN.2100	STARSIM.FN.2121, STARSIM.FN.2122, STARSIM.FN.2123
STARSIM.FN.2121	The electrical parameters shall be determined, if relevant for the simulation	STARSIM.FN.2120	
STARSIM.FN.2122	The thermal parameters shall be determined, if relevant for the simulation	STARSIM.FN.2120	
STARSIM.FN.2123	The mechanical parameters shall be determined, if relevant for the simulation	STARSIM.FN.2120	
STARSIM.FN.2130	GNC parameters and configurations shall be defined or estimated	STARSIM.FN.2100	STARSIM.FN.2131, STARSIM.FN.2132
STARSIM.FN.2131	The GNC functions shall be determined	STARSIM.FN.2130	STARSIM.FN.2131.1, STARSIM.FN.2131.2, STARSIM.FN.2131.3
STARSIM.FN.2131.1	The navigation strategies shall be determined	STARSIM.FN.2131	
STARSIM.FN.2131.2	The guidance strategies shall be determined	STARSIM.FN.2131	
STARSIM.FN.2131.3	The control strategies shall be determined	STARSIM.FN.2131	
STARSIM.FN.2132	The GNC architecture shall be determined	STARSIM.FN.2130	STARSIM.FN.2132.1, STARSIM.FN.2132.2, STARSIM.FN.2132.3
STARSIM.FN.2132.1	The sensors shall be chosen	STARSIM.FN.2132	
STARSIM.FN.2132.2	The actuators shall be chosen	STARSIM.FN.2132	
STARSIM.FN.2132.3	The processors and electronic devices shall be chosen and their behavior reproduced	STARSIM.FN.2132	
STARSIM.FN.2200	StarSim shall model and simulate the system and the surrounding condition	STARSIM.FN.2000	STARSIM.FN.2210, STARSIM.FN.2220, STARSIM.FN.2230, STARSIM.FN.2240, STARSIM.FN.2250

STARSIM.FN.2210	StarSim shall support the configuration of different simulation architectures	STARSIM.FN.2200, STARSIM.TL.090, STARSIM.OP.100	STARSIM.FN.2211, STARSIM.FN.2212, STARSIM.FN.2213
STARSIM.FN.2211	Different type of simulation shall be settable	STARSIM.FN.2210	STARSIM.FN.2211.1, STARSIM.FN.2211.2, STARSIM.FN.2211.3, STARSIM.FN.2211.4, STARSIM.FN.2211.5
STARSIM.FN.2211.1	AIL simulation configuration shall be settable	STARSIM.FN.2211, STARSIM.CNF.200	
STARSIM.FN.2211.2	SIL simulation configuration shall be settable	STARSIM.FN.2211, STARSIM.CNF.300	
STARSIM.FN.2211.3	CIL simulation configuration shall be settable	STARSIM.FN.2211, STARSIM.CNF.400	
STARSIM.FN.2211.4	HIL simulation configuration shall be settable	STARSIM.FN.2211, STARSIM.CNF.500	
STARSIM.FN.2211.5	Hybrid configuration among AIL,SIL,CIL,HIL shall be possible to built	STARSIM.FN.2211, STARSIM.CNF.700	
STARSIM.FN.2212	The StarSim shall know interfaces and connections among the simulation elements	STARSIM.FN.2210, STARSIM.OP.300	STARSIM.FN.2212.1, STARSIM.FN.2212.2, STARSIM.FN.2212.3
STARSIM.FN.2212.1	Information shall be formatted according protocols	STARSIM.FN.2212	
STARSIM.FN.2212.2	Software interface shall be set	STARSIM.FN.2212	
STARSIM.FN.2212.3	Hardware interface shall be set	STARSIM.FN.2212	
STARSIM.FN.2213	StarSim shall know the GSE involved in the simulation	STARSIM.FN.2210	
STARSIM.FN.2220	StarSim shall allow the selection of models	STARSIM.FN.2200	STARSIM.FN.2221, STARSIM.FN.2222, STARSIM.FN.2223
STARSIM.FN.2221	The type of model (virtual vs physical) shall be selected	STARSIM.FN.2220	STARSIM.FN.2221.1, STARSIM.FN.2221.2
STARSIM.FN.2221.1	The fidelity level shall be individuated for each virtual model	STARSIM.FN.2221	
STARSIM.FN.2221.2	The physical model details shall be specified	STARSIM.FN.2221	

STARSIM.FN.2222	The environmental and S/C motion models shall be specified	STARSIM.FN.2220, STARSIM.OP.210	STARSIM.FN.2222.1, STARSIM.FN.2222.2, STARSIM.FN.2222.3
STARSIM.FN.2222.1	The environmental and S/C motion models shall be selected from the databases	STARSIM.FN.2222	
STARSIM.FN.2222.2	Facilities, tools, and items shall be selected for each type of configuration	STARSIM.FN.2222	
STARSIM.FN.2222.3	The virtual environmental & S/C motion models shall be contained in \Environmental & SC motion\ folder.	STARSIM.FN.2222	
STARSIM.FN.2223	The equipment/devices, subsystem and system models shall be selected accordingly the design and the product life cycle phase	STARSIM.FN.2220	STARSIM.FN.2223.1, STARSIM.FN.2223.2
STARSIM.FN.2223.1	The devices and equipment models shall be selected from the databases folder "Devices&equipment".	STARSIM.FN.2223	
STARSIM.FN.2223.2	The MU, EM, QM, FM shall be selected accordingly to the selected simulation configuration	STARSIM.FN.2223	
STARSIM.FN.2230	StarSim shall set-up the simulation according to the user choices	STARSIM.FN.2200	STARSIM.FN.2231, STARSIM.FN.2232, STARSIM.FN.2233, STARSIM.FN.2234
STARSIM.FN.2231	The simulation parameters and conditions shall be set-up	STARSIM.FN.2230	
STARSIM.FN.2232	The interfaces (hardware and software) shall be set-up	STARSIM.FN.2230	
STARSIM.FN.2233	The kernel and firmware for any process shall be configured	STARSIM.FN.2230	
STARSIM.FN.2234	GSE shall be set-up	STARSIM.FN.2230	STARSIM.FN.2234.1, STARSIM.FN.2234.2, STARSIM.FN.2234.3
STARSIM.FN.2234.1	GSE parameters shall be initialized: starting condition shall be setup	STARSIM.FN.2234	
STARSIM.FN.2234.2	GSE functions and parts shall be disenabled/enabled, switched on/off and tuned	STARSIM.FN.2234	

STARSIM.FN.2234.3	Each model shall be setup in terms of constant parameter and variable initial values	STARSIM.FN.2234	
STARSIM.FN.2240	StarSim shall run with the chosen architecture and settings	STARSIM.FN.2200, STARSIM.OP.500	STARSIM.FN.2241, STARSIM.FN.2242
STARSIM.FN.2241	StarSim shall calculate the state variables for every simulation iteration	STARSIM.FN.2240	
STARSIM.FN.2242	StarSim shall update the outputs according to a specified frequency	STARSIM.FN.2240	
STARSIM.FN.2250	StarSim shall manage configurations, settings, and data information	STARSIM.FN.2200, STARSIM.OP.600, STARSIM.OP.400	STARSIM.FN.2251, STARSIM.FN.2252, STARSIM.FN.2253, STARSIM.FN.2254
STARSIM.FN.2251	The log files shall be properly opened, written/read and closed	STARSIM.FN.2250	
STARSIM.FN.2252	StarSim shall save and store sensible output	STARSIM.FN.2250	
STARSIM.FN.2253	StarSim shall save, store and restore simulation configurations and its parameters	STARSIM.FN.2250	
STARSIM.FN.2254	StarSim shall organize the data according to the choices of the user	STARSIM.FN.2250	STARSIM.FN.1110
STARSIM.FN.2300	StarSim shall support the user in the (GNC) system requirements verification and validation	STARSIM.FN.2000, STARSIM.TL.050, STARSIM.OP.700,	STARSIM.FN.2310, STARSIM.FN.2320
STARSIM.FN.2310	The data and parameters needed for the V&V shall be evaluated through the StarSim tools	STARSIM.FN.2300	STARSIM.FN.2311, STARSIM.FN.2312, STARSIM.FN.2313, STARSIM.FN.2314
STARSIM.FN.2311	StarSim shall import saved/stored results of each simulation session	STARSIM.FN.2310	
STARSIM.FN.2312	StarSim shall visualize data in terms of tables, plots, and graphs	STARSIM.FN.2310	
STARSIM.FN.2313	StarSim shall handle simulation data	STARSIM.FN.2310	
STARSIM.FN.2314	StarSim shall support the analysis of simulation data through dedicated input	STARSIM.FN.2310	
STARSIM.FN.2320	StarSim shall produce output usable for reports and documentation	STARSIM.FN.2300	

STARSIM.FN.2400	StarSim shall manage time and synchronization	STARSIM.FN.2000	STARSIM.FN.2410, STARSIM.FN.2420, STARSIM.FN.2430, STARSIM.FN.2440
STARSIM.FN.2410	StarSim shall hold the simulation time and CET time (or derived from it)	STARSIM.FN.2400	
STARSIM.FN.2420	The time tagging shall be added to any result and operation activity	STARSIM.FN.2400	
STARSIM.FN.2430	StarSim shall manage the time during both the RT and not-RT simulation session	STARSIM.FN.2400	STARSIM.FN.2431, STARSIM.FN.2432
STARSIM.FN.2431	StarSim shall guarantee RT execution with an error less than 10 ms (TBC)	STARSIM.FN.2430, STARSIM.TL.100	
STARSIM.FN.2432	StarSim shall adapt the time management in non-RT in order to maintain the time-step but execute the simulation as fast as possible	STARSIM.FN.2430, STARSIM.TL.100, STARSIM.TL.110	
STARSIM.FN.2440	StarSim shall synchronize the process: it adjust time operation execution wrt the time required by the StarSim parameters	STARSIM.FN.2400	
STARSIM.FN.2500	StarSim shall manage models and interface database	STARSIM.FN.2000, STARSIM.TL.090	STARSIM.FN.2510, STARSIM.FN.2520, STARSIM.FN.2530
STARSIM.FN.2510	StarSim shall organize the models in categories	STARSIM.FN.2500	STARSIM.FN.2511
STARSIM.FN.2511	The database shall be devided according parameters and features	STARSIM.FN.2510, STARSIM.CNF.610 , STARSIM.CNF.630	
STARSIM.FN.2520	StarSim shall accept new models, if built according to the specific rules of the StarSim	STARSIM.FN.2500, STARSIM.CNF.620	STARSIM.FN.2521
STARSIM.FN.2521	StarSim shall favor the insertion of new model into the database	STARSIM.FN.2520, STARSIM.TL.120	
STARSIM.FN.2530	StarSim shall help the physical and hybrid models characterization and calibration	STARSIM.FN.2500	
STARSIM.FN.2600	StarSim shall check the consistency of the defined configuration	STARSIM.FN.2000	STARSIM.FN.2610, STARSIM.FN.2620, STARSIM.FN.2630, STARSIM.FN.2640

STARSIM.FN.2610	StarSim shall check the software consistency of every generated software	STARSIM.FN.2600	
STARSIM.FN.2620	StarSim shall check the consistency of inserted/selected data in each model	STARSIM.FN.2600	
STARSIM.FN.2630	StarSim shall verify the correctness of the setup values for each interface	STARSIM.FN.2600	
STARSIM.FN.2640	StarSim shall check that all the initializations of interfaces, models and GSE are successfully completed	STARSIM.FN.2600	
STARSIM.FN.2650	StarSim shall individuate errors generated by bad set-up or models and interface choices made by the user	STARSIM.FN.3000	
STARSIM.FN.3000	StarSim shall generate code autonomously	STARSIM.FN.2600	
STARSIM.FN.3100	StarSim shall generate the software code skeleton	STARSIM.FN.3000	STARSIM.FN.3100, STARSIM.FN.3200
STARSIM.FN.3110	StarSim shall provide specific code libraries	STARSIM.FN.3100	
STARSIM.FN.3120	StarSim shall provide basic functions database	STARSIM.FN.3100	
STARSIM.FN.3130	StarSim shall generate source code skeleton (in C/C++)	STARSIM.FN.3100	
STARSIM.FN.3140	StarSim shall generate software events	STARSIM.FN.3100	STARSIM.FN.3141, STARSIM.FN.3142, STARSIM.FN.3143, STARSIM.FN.3144
STARSIM.FN.3141	StarSim shall create the instances for hardware and software interfaces	STARSIM.FN.3140	
STARSIM.FN.3142	StarSim shall manage the tasks through the scheduler	STARSIM.FN.3140	
STARSIM.FN.3143	StarSim shall generate pipes interface	STARSIM.FN.3140	
STARSIM.FN.3144	StarSim shall manage communication with I/O ports	STARSIM.FN.3140	STARSIM.FN.3144.1
STARSIM.FN.3144.1	The communication with I/O ports shall be setup following the specific parameters	STARSIM.FN.3144	
STARSIM.FN.3150	StarSim shall compile the code	STARSIM.FN.3100	
STARSIM.FN.3160	StarSim shall generate executable files for every kind of simulation configuration	STARSIM.FN.3100	
STARSIM.FN.3170	The code for embedded systems shall be generated through the StarSim tools	STARSIM.FN.3100	

STARSIM.FN.3200	StarSim shall provide the instructions for the code generation	STARSIM.FN.3000	STARSIM.FN.3210, STARSIM.FN.3220, STARSIM.FN.3230
STARSIM.FN.3210	StarSim shall provide user friendly interface and support functions	STARSIM.FN.3200	
STARSIM.FN.3230	StarSim shall individuate the compiling errors or code bugs before the executable file generation	STARSIM.FN.3200	STARSIM.FN.3231
STARSIM.FN.3231	A list of errors/bug shall be provided	STARSIM.FN.3230	
STARSIM.FN.3300	StarSim shall generate the onboard software code	STARSIM.FN.3000	STARSIM.FN.3310
STARSIM.FN.3310	StarSim shall provide specific code libraries for crosscompiling code	STARSIM.FN.3300	
STARSIM.FN.3320	StarSim shall generate source code skeleton for crosscompiling code	STARSIM.FN.3300	
STARSIM.FN.3330	StarSim shall cross-compile the code	STARSIM.FN.3300	STARSIM.FN.3331
STARSIM.FN.3331	StarSim shall provide the link to the cross-compiler	STARSIM.FN.3330	
STARSIM.FN.3340	StarSim shall generate executable code for the specific embedded application	STARSIM.FN.3300	
<i>Operational requirements</i>			
STARSIM.OP.100	The user shall select the simulation architecture clicking the button "DEFINE SIMULATION ARCHITECTURE" on the main GUI window		STARSIM.OP.110, STARSIM.OP.120, STARSIM.OP.130
STARSIM.OP.110	The user shall define the number of processes involved in the simulation session setting up the number on the PROCESS SELECTION pop up and after pushing OK	STARSIM.OP.100	
STARSIM.OP.120	The user shall define for any process the parameters: tempistica, priority, path and interface (as spacificed in the STESINA PhD thesis - paragraph 3.3)	STARSIM.OP.100	STARSIM.OP.121, STARSIM.OP.122, STARSIM.OP.123, STARSIM.OP.124, STARSIM.OP.125, STARSIM.OP.126, STARSIM.OP.127, STARSIM.OP.128, STARSIM.OP.129

STARSIM.OP.121	The interface for any process shall be setup in terms of type, name, path and speed (as spacificed in the STESINA PhD thesis - paragraph 3.3)	STARSIM.OP.120	
STARSIM.OP.122	The user shall set the interfaces (hardware and software)	STARSIM.OP.120	
STARSIM.OP.123	The user shall set named pipes through the GUI following the rules in STESINA PhD thesis	STARSIM.OP.120	STARSIM.FN.3143
STARSIM.OP.124	The user shall set the serial ports parameters through the GUI following the rules in STESINA PhD thesis	STARSIM.OP.120	STARSIM.FN.3144
STARSIM.OP.125	The user shall set-up the USB ports parameters through the GUI following the rules in STESINA PhD thesis	STARSIM.OP.120	STARSIM.FN.3144
STARSIM.OP.126	The user shall set-up the LAN ports parameters through the GUI following the rules in STESINA PhD thesis	STARSIM.OP.120	STARSIM.FN.3144
STARSIM.OP.127	The user shall set-up the interfaces protocols	STARSIM.OP.120	STARSIM.OP.321, STARSIM.OP.322
STARSIM.OP.128	The user shall define the "data field" of the selected high level protocol	STARSIM.OP.120	
STARSIM.OP.129	The user shall define the communication protocols used by sensors, actuators, devices, etc... involved in the simulation (in particular for CIL and HIL configurations)	STARSIM.OP.120	
STARSIM.OP.130	The user shall save the current session setup pushing SAVE on the PROCESS PROPERTIES window	STARSIM.OP.100	
STARSIM.OP.200	The user shall choose the models from the database	STARSIM.TL.090	STARSIM.OP.201, STARSIM.OP.202, STARSIM.OP.203, STARSIM.OP.204
STARSIM.OP.201	The user shall surf in the model database through the MODELS SELECTION window (left part)	STARSIM.OP.200	
STARSIM.OP.202	Database structure is available in the STESINA PhD thesis	STARSIM.OP.200	
STARSIM.OP.203	The user shall add the chosen models pushing ADD button	STARSIM.OP.200	
STARSIM.OP.204	The user shall remove the chosen models pushing REMOVE button	STARSIM.OP.200	
STARSIM.OP.250	The user shall set-up the parameters (variables and constants) for each selected model through the GUI window MODEL PARAMETERS	STARSIM.OP.200	

STARSIM.OP.300	The user shall setup the simulation parameters through the pop up presented after pushing the button SIMULATION PARAMETERS		STARSIM.OP.310
STARSIM.OP.310	The user shall explain a value for any required parameter (however default values are present)	STARSIM.OP.300	
STARSIM.OP.400	The user shall setup the outputs files containing the chosen output parameters		STARSIM.OP.410, STARSIM.OP.420
STARSIM.OP.410	The user shall select the sensible output data ticking the relative little square in the OUTPUT SELECTION window	STARSIM.OP.400	
STARSIM.OP.420	The user shall specify the file name and path in the StarSim in which the sensible outputs are saved during the execution	STARSIM.OP.400	
STARSIM.OP.500	The user shall manage, monitor and control the simulation execution		STARSIM.OP.501, STARSIM.OP.503, STARSIM.OP.503
STARSIM.OP.501	The user shall start the simulation session, pushing the button "GO"	STARSIM.OP.500	
STARSIM.OP.502	The user shall stop the simulation session, pushing the button "STOP"	STARSIM.OP.500	
STARSIM.OP.600	The user shall require the StarSim outputs of current or previous simulation sessions: the button "ACCESS DATA" recall the data repository		STARSIM.OP.610
STARSIM.OP.610	StarSim shall recovery the information from data repository	STARSIM.OP.600	
STARSIM.OP.700	The user shall require the StarSim configuration of current or previous simulation sessions: the ACCESS DATA button shall recall the configuraion from the data repository		STARSIM.OP.710
STARSIM.OP.710	StarSim shall recovery the configuration settings from data repository	STARSIM.OP.700	
STARSIM.OP.800	The user shall update the models and interfaces databases	STARSIM.TL.120	STARSIM.CNF.620
<i>Configuration Requirements</i>			
STARSIM.CNF.100	The StarSim shall be constituted by a Simulation Unit, a Control Console, Ground Support Equipment, a Test Object.		STARSIM.CNF.110

STARSIM.CNF.110	The Simulation Unit shall have a core and an interface unit	STARSIM.CNF.100	STARSIM.CNF.111 STARSIM.CNF.112S TARSIM.CNF.113 STARSIM.CNF.114S TARSIM.CNF.115
STARSIM.CNF.111	The core of the Simulation Unit shall host the software to manage the configurations settings decided by the user	STARSIM.CNF.110	
STARSIM.CNF.112	The core of the Simulation Unit shall host the software devoted to manage the simulation execution	STARSIM.CNF.110	
STARSIM.CNF.113	The simulation core shall have the memory allocated to store data repository	STARSIM.CNF.110	
STARSIM.CNF.114	The interface unit shall manage hardware and software interfaces	STARSIM.CNF.110	STARSIM.INT.100, STARSIM.INT.300, STARSIM.INT.400, STARSIM.INT.500, STARSIM.INT.630
STARSIM.CNF.120	The Control Console shall have a core and an interface unit	STARSIM.CNF.100	STARSIM.CNF.121 STARSIM.CNF.122S TARSIM.CNF.123
STARSIM.CNF.121	The core of the Control Console shall host the software to manage the GUI	STARSIM.CNF.120	
STARSIM.CNF.122	The GUI shall allow the interaction with the user	STARSIM.CNF.120	
STARSIM.CNF.130	The Test Object could be represented by sensors, actuators and embedded systems	STARSIM.CNF.100	STARSIM.INT.600
STARSIM.CNF.140	The GSE shall be configurated in terms of parameters setup and physical arrangement from the user or the Simulation Unit	STARSIM.CNF.100	STARSIM.CNF.141 STARSIM.CNF.142 STARSIM.INT.300
STARSIM.CNF.141	Power frontends shall be set with the desired voltage and current values	STARSIM.CNF.140	
STARSIM.CNF.142	Power frontend for regulation shall provide regulation from the 220V to 3.3, 5, 9, 12, 28 Volt with a limited current	STARSIM.CNF.140	
STARSIM.CNF.146	Communication frontend shall guarantee the communication through serial,USB,LAN, ... , also adapting the logic level	STARSIM.CNF.140	

STARSIM.CNF.200	The AIL configuration shall be made according the document [Stesina PhD Thesis] - pag 47		STARSIM.CNF.201 STARSIM.FN.2111.1,
STARSIM.CNF.201	More than one simulation session should be possible to perform in AIL configurations	STARSIM.CNF.200, STARSIM.TL.110	
STARSIM.CNF.300	The SIL configuration shall be made according the document [Stesina PhD Thesis] - pag.48		STARSIM.CNF.301, STARSIM.FN.2111.2
STARSIM.CNF.301	More than one simulation session should be possible to perform in SIL configurations	STARSIM.CNF.300, STARSIM.TL.110	
STARSIM.CNF.400	The CIL configuration shall be made according the document [Stesina PhD Thesis] - pag. 48		STARSIM.FN.2111.3
STARSIM.CNF.500	The HIL configuration shall be made according the document [Stesina PhD Thesis] - pag. 49		STARSIM.FN.2111.4
STARSIM.CNF.600	StarSim shall have a models database in its non volatile memory	STARSIM.CNF.115	STARSIM.CNF.610, STARSIM.CNF.620, STARSIM.CNF.630
STARSIM.CNF.601	StarSim shall have instruments to manage, store and update data models	STARSIM.CNF.110, STARSIM.OP.800, STARSIM.CNF.600	
STARSIM.CNF.610	The models database shall be organized in the following categories: 1.Equiment and devices models, 2. Mission models, 3. GSE models, 4. Special Funtions, 5. GNC strategies, 6. Transformation & Conversion, 7. Visualize and display	STARSIM.CNF.600	
STARSIM.CNF.620	Each model shall be stored in a C/C++ function	STARSIM.CNF.600	STARSIM.CNF.621 STARSIM.CNF.622 STARSIM.CNF.623
STARSIM.CNF.621	The C++ function for model shall be generated inserting first all the input models variables and then the output model variables	STARSIM.CNF.620	
STARSIM.CNF.622	The configurable parameters for each model shall be indicated within an header and a cloder marker. The header marker is /*CONFIGURABLE SESSION*/ and the closer marker is /*END-CONFIG*/	STARSIM.CNF.620	
STARSIM.CNF.623	The configurable parameters for each model shall be connected to a GUI window	STARSIM.CNF.620	

STARSIM.CNF.630	The interfaces database shall be organized in the following categories: 1.Hardware interfaces, 2. Software interfaces, 3. Protocols Management	STARSIM.CNF.600	STARSIM.CNF.631, STARSIM.CNF.632, STARSIM.CNF.633
STARSIM.CNF.631	The hardware interface shall be configured according to the [Stesina PhD Thesis pag 128]	STARSIM.CNF.630	
STARSIM.CNF.632	The software interface shall be configured according to the [Stesina PhD Thesis pag 128]	STARSIM.CNF.630	
STARSIM.CNF.633	The protocol management shall be made following the instruction to the [Stesina PhD Thesis pag 128]	STARSIM.CNF.630	
STARSIM.CNF.640	Hybrid configuration among AIL,SIL,CIL,HIL shall be settable	STARSIM.CNF.600	
STARSIM.CNF.1000	The workstation shall guarantee to perform simulation session for any acceptable configuration		
STARSIM.CNF.1001	The workstation shall have at least 12 processors	STARSIM.CNF.1000	
STARSIM.CNF.1002	The workstation shall have at least 16 Gbyte RAM memory available	STARSIM.CNF.1000	
STARSIM.CNF.1003	The workstation shall have 2000 Gbyte of Harddisk memory	STARSIM.CNF.1000	
STARSIM.CNF.1004	The workstation shall have at least the following I/O peripherals: 3 serial ports, 12 USB ports, the ethernet port (TBC), 2 video output	STARSIM.CNF.1000	
STARSIM.CNF.1005	The workstation shall have keyboard and mouse	STARSIM.CNF.1000	
STARSIM.CNF.1100	The workstation shall have the Linux release 13.10	STARSIM.DS.110	
STARSIM.CNF.1101	The workstation shall have a Linux kernel 13.10_RT devoted to the Real Time operations	STARSIM.DS.111	
<i>Interface Requirements</i>			
STARSIM.INT.100	The Simulation Unit and the Control Console shall communicate through a dedicated software process and/or via LAN		
STARSIM.INT.200	The Control Console and the user shall communicate through the GUI		STARSIM.INT.201
STARSIM.INT.201	The user shall interact with the Control Console using the mouse and the keyboard	STARSIM.INT.200	
STARSIM.INT.300	The GSE and the Simulation Unit shall be interfaced through the interface unit of the simulation unit		STARSIM.INT.310, STARSIM.INT.320
STARSIM.INT.310	The Simulation Unit and the GSE shall communicate through hardware interfaces (e.g. serial, USB,LAN, CAN, I2C) and specific, custom protocols defined in datasheet	STARSIM.INT.300	

STARSIM.INT.320	The Simulation Unit and the GSE shall communicate information according to a specific or custom protocol setted in the relative files contained in the folder "Manuals and Datasheets"	STARSIM.INT.300	
STARSIM.INT.400	The StarSim core shall communicate with the virtual models (mission or equipment) through processes (managed by the Simulation Unit - Interface Unit)		
STARSIM.INT.500	The Test Object equipment and embedded system shall interface according to the specific design of the Test Object		
<i>Design Requirements</i>			
STARSIM.DS.100	The StarSim programs shall run on a workstation or a network of computers		STARSIM.DS.110, STARSIM.DS.120, STARSIM.DS.130
STARSIM.DS.110	The workstation shall have Linux as OS and RTOS	STARSIM.DS.100	STARSIM.DS.111, STARSIM.DS.112, STARSIM.CNF.1100
STARSIM.DS.111	The workstation shall have a Linux kernel devoted to the Real Time operations	STARSIM.DS.110	STARSIM.CNF.1101
STARSIM.DS.112	The workstation shall have Linux kernel devoted to the non-RT operations	STARSIM.DS.110	STARSIM.CNF.1102
STARSIM.DS.120	The StarSim software shall be written in C++ and Python	STARSIM.DS.100, STARSIM.TL.040, STARSIM.TL.070	STARSIM.DS.121, STARSIM.DS.122, STARSIM.DS.123, STARSIM.DS.128
STARSIM.DS.121	The simulation software for the simulation management shall be written in Python	STARSIM.DS.120	
STARSIM.DS.122	The GUI shall be developed and updated using Python	STARSIM.DS.120, STARSIM.TL.070, STARSIM.PL.040	
STARSIM.DS.123	The models and interfaces code shall be developed in C/C++	STARSIM.DS.120, STARSIM.TL.070, STARSIM.PL.040	
STARSIM.DS.124	A Python program shall manage the database	STARSIM.DS.120	

STARSIM.DS.130	The workstation shall be able to support every type of simulation operations and StarSim activity	STARSIM.DS.100, STARSIM.FN.1000, STARSIM.FN.2000, STARSIM.FN.3000	STARSIM.DS.131, STARSIM.DS.132, STARSIM.DS.133, STARSIM.DS.134, STARSIM.DS.135, STARSIM.DS.136
STARSIM.DS.140	The workstation shall host Eclipse as software development environment	STARSIM.DS.100	
STARSIM.DS.200	Power GSE shall provide regulated voltages and currents	STARSIM.CFN.142	
STARSIM.DS.300	Communication GSE shall allow the hardware communication for the most common high level communication protocols	STARSIM.CFN.146	
STARSIM.DS.400	GSE emulation sensors output shall provide the correct stimulation both from electrical, logical and mechanical point of view	STARSIM.CFN.140	

Table 24: StarSim's list of requirements

3.3 Simulator architecture

The Figure 51 shows the top top level architecture of the simulator where five main actors are individuated and described in the next paragraphs: the test object, the simulation unit, the Ground Support Equipment, and the Control Console.

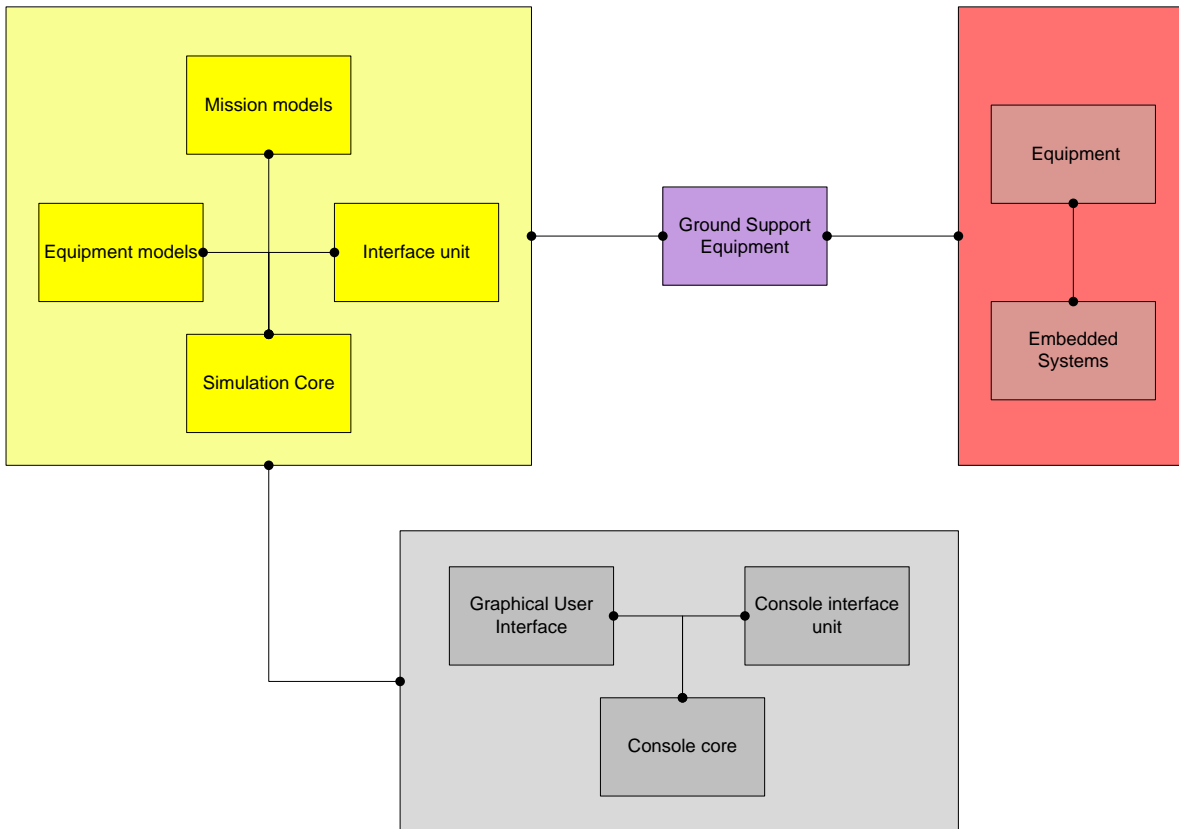


Figure 51: General simulator's architecture

3.3.1 The Simulation Unit

The Simulation Unit is the main element of the simulator and it is constituted by:

- the simulation core,
- the interface unit,
- the environmental models and the equipment models.

This part of the StarSim as well as the Control Console and Interface Unit has been designed and implemented with Eng. Lorenzo Feruglio (STAR Team of Politecnico di Torino).

3.3.1.1 Simulator core

The core [5] is constituted by a WorkStation (WS) with Linux as Operating System (OS). Multi-core processors are involved and work in parallel. Details on the WS hardware are in the Appendix F. For the choice of the Hardware of WorkStation, some issues and obstacles have been met:

- The limited money resources available for the research;
- The red tape of the Italian public administration: that requires a slow and unskilled intrusion of its technicians and muddled procedures for the acquisition;
- The limited lifetime of hardware architectures: processor families typically are around for only a few years before the next generation supersedes them. Simulation software must often be upgraded or rewritten to remain workable.

- The improved performance at lower prices: The cost of a unit of computing power decreases by a factor of 10 every 3 years. Many applications that require intensive computing can now run well on inexpensive desktop computers.

In the release v1.x, all the Simulation Unit software are executed by the WS. Software developed for multiple and parallel processors (a distributed architecture) can be difficult to write and it is particularly hard to debug. These difficulties are limited from the use of software developing code environments (i.e. Eclipse) that can easy manage multi-flows operations and provide user friendly tools helping the developer.

Two software have been developed for StarSim: ones has management tasks, others is the core of the simulation activities.

The first program is written in Python and it is able to: 1) setup the simulation 2) manage the logical interfaces with the *test-object*, 3) manage the I/O with the user through the Control Console (display, keyboard and mouse), 4) schedule all tasks according to priorities and time and logical sequence during the execution, and 5) save all the information related to the simulation session.

The WS operates as the simulator core when a C++ program, supported by a Python code for the management functions, runs. All the unnecessary functionalities of the OS are inhibited in order to guarantee as much as possible the real time activity of the simulator. *Ad hoc* kernels are defined to allow different types of simulations (real-time or not real-time) for several arrangements of the test object hardware. The C++ program 1) contains all the custom developed functions and header files needed to perform the simulation, 2) simulates the behavior of the virtual models, 3) manages the interface with external HW elements, 4) manage time synchronization and data, and 5) save information in files for post processing activities.

3.3.1.1.1 Process vs. Class vs. Threads

This paragraph presents and discusses the elements at the base of base a high performance code generation.

The main issue is to implement an architecture that relies on parallelisms. It concerns the possibility of executing “at the same time” different instances of code that execute different functions. Different levels of parallelism can be reached; it depends on the available hardware:

- For single processor architectures, the operation flux will be something linear, as the instruction flow will be handled by a single processor: in theory, two instructions cannot be processed at the same time. What is usually done in this type of machines is the instantiation of *processes* or *threads* that allow differentiating between several instruction flows at a logic level, keeping, however, a single instruction flow at the processor.
- For multi-processor architectures, it will be possible to sort the different processes or threads between the various processors, obtaining in this case an execution truly in parallel of instructions. The hardware capability will then set the limit for the number of processes/threads that can be run on every processor (depending as well on the weight of the processes).

The main difference between a *process* and a *thread* lies in the assigned or exploited resources: a thread is handled in the memory space already assigned to the spawning process, while a process is spawned by the OS, on code demand, in a new memory area.

An overview of the difference between threads and processes is now proposed.

A thread is characterized by these features:

- It can be considered as a “lightweight” process;
- Its execution state can take different values (*running, halted, ready, etc*);

- It can be prioritized by either the OS and the code itself;
- It utilizes resources already allocated to the calling process.

A process can be defined as a running instance of a program. This definition already comes with a few concepts included: a program is not what runs (it resides on a non-volatile media); what runs is its instance that is executed with at least some portions in the RAM of a machine

A process is characterized by these features:

- A dedicated resource space
- A protected access to other processors, processes or I/O resources.

Processes can share resource space in the event of a copy-on-write implementation: an optimization strategy used in case of multiple callers asking for resources initially indistinguishable: they can all be given pointers to the same resource. This can be maintained until a caller tries to modify its "copy" of the resource, at which point a separate (private) copy is made for that caller to prevent its changes from becoming visible to everyone else. The primary advantage is that if no caller ever makes any modifications, no private copy need ever be created.

It is possible to implement an architecture single/multi process, and it is possible to instantiate one or more threads in every process.

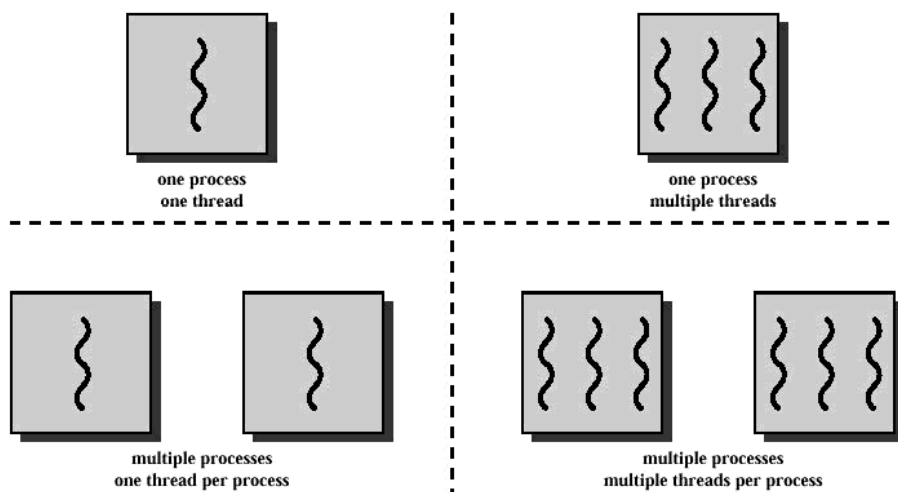


Figure 52: Processes and Threads

The implementation of processes and threads has its own drawbacks and advantages and the decision of the code structure is not trivial. [4]

3.3.1.1.2 Processes: fork(), execve() functions

All processes share a few characteristics among them: every process has a parent which created it (except the first process); similarly, a process is created as a child process, with the process responsible for its creation being its parent; a process can have only a single owner at a given time (ownership can be changed but it is exclusive).

Fork() function is a very important and unique characteristic of UNIX architecture systems (therefore a code written utilizing this function will be executable only on similar architecture platforms) and it allows the creation of a child process from a *father* that spawns it. The created child is a copy of the parent process, and the two different processes are differentiated by a few features. The function returns value 0 in the child process, while in the father process it returns an integer that is the Process Indicator of the child process. Must be remembered that Process Indicator s are unique and there

cannot be two processes with the same Process Indicator. It will return a value of -1 in case of error, no child process will be created and stderr will be set to the appropriate error code. The differences between child and father are:

- Child does not inherits father's memory blocks
- Resources usage of the process and its CPU time is set to zero
- Incoming signal list is emptied
- Father's semaphore' modifications are not inherited by the child
- Child process does not receive any signal for father termination; on the other side, the signal sent to the father for child termination is always SIGCHILD
- Other less common functions, not important for this work

Each other father feature is heritage by the child.

An interesting characteristic of forking derives from the possibility to call it just before calling the *exec()* function (or one of its variants, i.e. *execve()*). In this way after a check on Process Indicator two executable run (ones on the father and ones on the child), obtaining the creation of a new process that is running another program and not just a copy of the father code.

This is an extremely important features and one of the actual must-have parts for this simulator technology, as it boosts modularity and ease of code management (having to compile the different executables in other projects, reducing this way the dimensions of the simulator code).

```
pid = fork();
if (pid == -1) {
    printf ("Error!\n");
    return (0);
}
if (pid == 0) {
    ... child's code ...
}
else {
    ... father's code ...
}
```

In particular, the calling of *exec()* function in the child is performed in this way:

```
if (pid == 0) {
    exec ();
}
```

What is actually called in this simulator is the *execve()* function, that is what lies behind the several front-ends that are the *exec()* (and variations) functions.

The prototype is as follows: *int execve(const char *filename, char *constargv[], char *constenvp[])*; *Execve()* executes the program pointed to by *filename*. It must be either a binary executable, or a script starting with a line of the form: *#! interpreter [optional-arg]*, where *argv* is an array of argument strings passed to the new program. By convention, the first of these strings should contain the filename associated with the file being executed. *envp* is an array of strings, conventionally of the form **key=value**, which are passed as environment to the new program. Both *argv* and *envp* must be terminated by a NULL pointer. The argument vector and environment can be accessed by the called program's main function, when it is defined as explained before. *Execve()* does not return on success, as on success it will be launched a pre-compiled code.

In order to program an efficient and performing code it is interesting to compare performances of threads and processes when set up to do similar tasks, including in the comparison other parameters, such as simplicity of configuration, ease of debugging, resources taken, and so on. It must be noticed that such comparison is architecture-dependent, as processes on Microsoft Windows OS are handled in

a whole different way, resulting in different performances for process management when compared to process management on UNIX.

Overall, the situation is in favor of threads in Microsoft OS since processes are heavy and use a lot of resources; in addition they even have slower starting up times.

However, since in an embedded application UNIX architectures have been chosen, a comparison between processes and threads will be given only for UNIX architectures. The little benefits of threads compared to processes are not worth the difficulties that arise in coding/debugging/management of them. This is due to the UNIX structure: in fact, it is designed to support filtering (small programs that directly chain the output from one process to the input of the next). That relies on command shells (or other programs) is able to launch multiple programs with relatively little overhead, and chain the outputs to the inputs. A `fork()/exec()` combination is the way that this is done (the command shell forks a copy of itself, and the new process overlays itself with the program to be executed). Accordingly, the basic architecture of UNIX is designed to ensure the overhead of `fork()` and `exec()` is relatively small (where with overhead is intended the excess of resources being needed for `fork()/exec()` commands execution compared to threads).

3.3.1.1.3 Signal Handling

The signal handling stays for the activities made by the scheduler to manage events (such as interrupts) that can occur both on the nominal execution flow and, in particular, among process. Those signals (independent from the lines) are connected to numbers (see Table 25) that correspond to events that should be managed.

From the software engineering point of view, a signal is an *interrupt* delivered to a process. The operating system uses signals to report exceptional situations to an executing program. Some signals report errors such as references to invalid memory addresses, others report asynchronous events, such as disconnection of a line. The GNU C library defines a variety of signal types, each for a particular kind of event. Some kinds of events make it inadvisable or impossible for the program to proceed as usual, and the corresponding signals normally abort the program. Other kinds of signals that report harmless events are ignored by default. If the event that causes signals can be anticipated, it is possible to define a handler function and tell the operating system to run it when that particular type of signal arrives. Finally one process can send a signal to another process, this allows a parent process to abort a child, or two related processes to communicate and synchronize.

Signal generation usually can be grouped into three categories: errors, external events and explicit requests. Error means that a program has done something invalid and cannot continue execution. Obviously not every error raises a signal, but only few of them do. An external event generally has to do with I/O or other processes. These include the arrival of input, the expiration of a timer, and the termination of a child process. Explicit requests, on the other side, mean the use of a library function such as *kill* (whose purpose is specifically to generate a signal).

Signal generation can be performed in two ways regarding timings: *synchronously* or *asynchronously*.

- A synchronous signal pertains to a specific action in the program, and is delivered (unless blocked) during that action. A great number of errors generate synchronous signals and make explicit requests to a process to generate a signal for that same process. Sometimes, any errors (usually floating-point exceptions) are not reported completely synchronously, but may arrive a few instructions later.
- Asynchronous signals are generated by events outside the control of the process that receives them. They arrive at unpredictable times during execution. External events generate signals asynchronously, and so do explicit requests that apply to some other process.

Signal handling is done in C basically in two ways: utilizing *signal* or *sigaction* functions. The latter are stronger, but for the purpose of this work the *signal* are sufficient. The *signal* function establishes

action as the action for the signal *signum*: *sighandler_t signal (intsignum, sighandler_t action)*, where *sighandler_t* is the type of the signal handler functions. For the purpose of catching an alarm firing in a process, the function can be called in this way: *signal(SIGALRM,function_to_be_executed)*, assuring the execution of the function given in the argument in the second place.

There is a second function, the *kill* function, used especially in the watchdog that has been used to send a signal to another process. Despite its name, it can be used for a lot of things other than causing a process to terminate, for example

- a parent process starts a child to perform a task (perhaps having the child running an infinite loop) and then terminates the child when the task is no longer needed – watchdog case, or even scheduler behavior
- a process executes as part of a group and needs to terminate or notify the other processes in the group when an error or other event occurs
- two processes need to synchronize while working together

Kill function (*int kill(pid_tpid, intsignum)*) is declared in *signal.h* and sends the signal *signum* to the process or process group specified by *pid*. Besides the standards signals, *signum* can also have a value of zero to check the validity of the *pid*.

The *pid* specifies the process or process group to receive the signal:

- *pid > 0*: the process whose identifier is *pid*
- *pid == 0*: all processes in the same process group as the sender
- *pid < -1*: the process group whose identifier is *-pid*
- *pid == -1*: if the process is privileged, sends the signal to all processes except for some special system processes. Otherwise, sends the signal to all processes with the same effective user ID.

A process can send a signal to itself with a call like *kill(getpid(), signum)*.

The return value of *kill* is zero if the signal is sent correctly, otherwise it returns -1 and no signal is sent. If *pid* specifies sending a signal to several processes, the function succeeds if it can send the signal to at least one of them and there is no way to tell which of the processes got the signal or whether all of them did (without proper implementation of signal handling).

The Single UNIX Specification explains the following signals which are defined in *signal.h* (Table 25)

Signal	Description	Signal number on Linux x86
SIGABRT	Process aborted	6
SIGALRM	Signal raised by alarm()	14
SIGBUS	Bus error: “access to undefined portion of memory object”	7
SIGCHLD	Child process terminated, stopped (or continued)	17
SIGCONT	Continue if stopped	18
SIGFPE	Floating point exception: “erroneous arithmetic operation”	8
SIGHUP	Hangup	1
SIGILL	Illegal instruction	4

SIGINT	Interrupt	2
SIGKILL	Kill (terminate immediately)	9
SIGPIPE	Write to pipe with no one reading	13
SIGQUIT	Quit and dump core	3
SIGSEGV	Segmentation violation	11
SIGSTOP	Stop executing temporarily	19
SIGTERM	Termination (request to terminate)	15
SIGTSTP	Terminal stop signal	20
SIGTTIN	Background process attempting to read from tty (“in”)	21
SIGTTOU	Background process attempting to read from tty (“out”)	22
SIGUSR1	User-defined 1	10
SIGUSR2	User-defined 2	12
SIGPOLL	Pollable event	29
SIGPROF	Profiling timer expired	27
SIGSYS	Bad syscall	31
SIGTRAP	Trace/breakpoint trap	5
SIGURG	Urgent data available on socket	23
SIGVTALRM	Signal raised by timer counting virtual time	26
SIGXCPU	CPU time limit exceeded	24
SIGXFSZ	File size limit exceeded	25

Table 25: Single UNIX Specification signals defined in *signal.h* [5]

Particular attention has been given to the *watchdog* functions because they are strategic in autonomous applications. Three are the signals utilized in the *watchdog* (6, 9, and 15), two of them are used during inter-process communication:

- 15 – this is the default signal sent by *kill*. The use of this signal allows applications to clean up (delete temporary files, free system resources like semaphores, etc) before terminating at the user’s request.
- 9 – the default action of this signal (termination) cannot be changed. This provides a sure-fire mean of stopping an otherwise unstoppable process.

3.3.1.1.4 Real Time Operating System and Kernel

One of the topics when a simulator is designed refers the possibility to execute tasks within specified time intervals, or meeting certain deadlines (or missing it by foreseeable amounts of time). UNIX (and usually every OS architecture) process execution relies on scheduling policies in order to grant the

correct operability of the system and the non-chaotic execution of processes, that includes allocation of resources and the relative de-allocation of them.

Scheduling policy on UNIX architecture is made by assigning a priority level to a defined process.

The main issue when developing real time applications on non real time capable OS is that the OS itself can hang the execution of user level applications to perform higher priority tasks or when prompted by hardware signals (hardware interrupts). This results in unpredictable behavior of a user level application when considering the fulfillment of the timing requirements: it is not guaranteed the on-time execution of a task and it is not possible to predict when a task will start or finish with a desired precision.

A key issue to develop and implement a hard real time capable simulator is to utilize a hard real time operative system. There are a few possibilities when choosing the RTOS to be installed, some of them are open source while for the great part of them a license shall be bought. In this work it will be shown how to apply a real time patch to a vanilla kernel (vanilla is the term used to define a clean Linux kernel). Although other methods are available, ease of installation points to the second option, the real time patch to the Linux kernel: this is even suggested because it is possible to utilize newer kernels which are better supported on newer machines. Another option can be RTLinux which is maintained only in the paid version, while the free one is not updated. As for RTLinux, the idea behind its functioning is simple and effective and it has been used in similar HIL simulation, whose articles can be found in literature.

As introduced, the idea behind RTLinux is interesting: real time capability is reached shifting one level higher the Linux kernel, making it run on top of a hard-real time micro kernel that will run the Linux kernel as a full pre-emptive process, set up to handle and catch the hardware interrupts, making it impossible for them to slow down previously set up real time modules.

RTLinux comes in two main versions: one open source and one for pay, which is developed by WindRiver and offers more support to users. For this work RTLinux free has been used, which has all the characteristics needed to support StarSim activities.

Anyway, the ideal and suggested way to install a RTOS (patch to a Linux vanilla kernel) is probably more robust and less prone to errors, but it is allowed for less configuration of the system. In the following lines the installation method of the vanilla kernel and its patching is presented:

```
#sudo apt-get install kernel-package fakeroot build-essential libncurses5-dev
```

This will install as usual the necessary packages to perform the OS installation.

Then, the vanilla kernel image and the relative real time patch are downloaded and extracted. Instruction are posted here for version 3.0.10, tweak the commands as necessary.

```
# mkdir -p ~/tmp/linux-rt  
# cd ~/tmp/linux-rt  
# wget http://www.kernel.org/pub/linux/kernel/v3.0/linux-3.0.10.tar.bz2  
# wget http://www.kernel.org/pub/linux/kernel/projects/rt/3.0/patch-3.0.10-rt27.patch.gz  
# tar xjvf linux-3.0.10.tar.bz2  
# gunzip patch-3.0.10-rt27.patch.gz
```

The kernel is now patched

```
# cd linux-3.0.10  
# patch -p1 < ../patch-3.0.10-rt27.patch
```

Kernel configuration is now made with

```
# cp /boot/config-$(uname -r) .config && make oldconfig
```

Full pre-emption (option 5) should be selected when prompted, and the rest should be left as it is. Full pre-emption sets the system in real time operation mode.

Kernel is then built with these commands

```
# sed -rie 's/echo "\+"/#echo "\+"/' scripts/setlocalversion
# make-kpkg clean
# CONCURRENCY_LEVEL=$((($(getconf _NPROCESSORS_ONLN)+1)) fakeroot
make-kpkg --initrd --revision=0 kernel_imagekernel_headers
```

This last command creates the initrd image needed and the various headers.

Lastly the kernel is installed with this command

```
# sudodpkg -i ../linux-{headers,image}-3.0.10-rt27_0_*.deb
```

It will be now possible to boot into the new OS. Depending on the configuration of the machine, a manual editing of the GRUB menu/file might be needed. Known issues are present in machines with NVidia Graphics Cards: further patching is needed to avoid problems.

The success of the installing operation can be checked under an UBUNTU OS opening System Monitor under System tag as shown in Figure 53

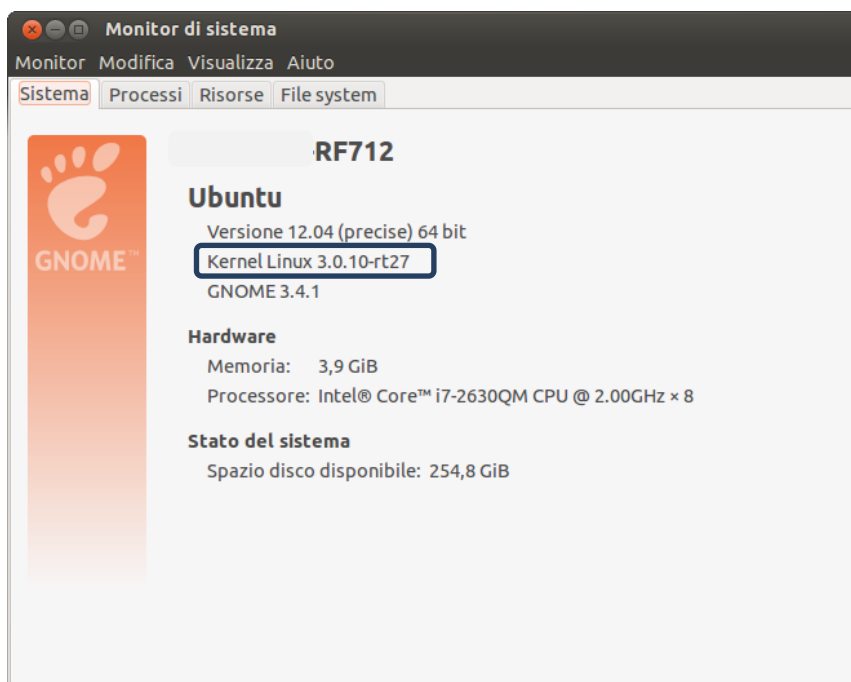


Figure 53: Windows of the successful installation of Linux RT-patched Kernel

3.3.1.1.5 Assigning real time capabilities to programs

Configuring and installing a RTOS is not sufficient: such kind of OS does not run every single process in real time, but real time execution must be enabled within the program code.

1. Main option to do in RTLinux environment is to create an object file (created by compiling in gcc with the `-c` flag an ordinary C language file without `main()` function). Instead of the main function there will be `init_module` and `cleanup_module` functions:
 - `Init_module()` function is called when the module is inserted into the kernel and registers a handler in the kernel;
 - `Cleanup_module()` is called before the module is removed and basically undoes what the `init_module()` function did.

After the module has been compiled with

```
# gcc -c (FLAGS) my_module.c
```

the module can be inserted into the kernel with the command

```
# insmodnameofmodule.o
```

while removing the module is performed with

```
# rmmoduleofmodule
```

2. For the second option, enabling RT capabilities for the process is done utilizing the `sched_setscheduler()`, and assigning a real time scheduling policy (which is now handled by the OS) and a priority (ranging from 1 – lowest to 99 – highest). Usually user processes can't change the priority level of a process and setting it to FIFO or RR, therefore what needs to be done is to compile the code through terminal (thus creating an executable file) and running that file with root privileges.

This is not a secure proceeding in regards of safety and exploitability of a process / program, but for now it is enough and serves the purpose.

Commands needed to compile the various files are:

```
# cc *.c -o main -lpthread -lrt
```

where the `-lpthread` flag is used to compile a program that utilizes POSIX threads.

The new executable file can now be launched by terminal with the command

```
# sudo ./main
```

The last procedure usually allows the execution of such processes in a *soft* real time way on non RTOS. The installation of a RTOS is needed to assure the *hard* real time execution mode.

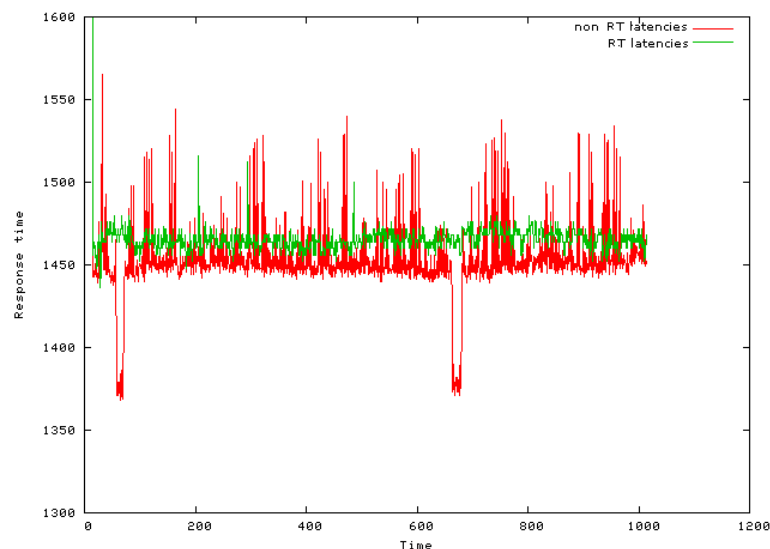


Figure 54: The importance of RTOS

The Figure 54 shows why it is important to execute simulation in a real time operating system: despite having overall slower performance, the real time OS guarantees that no latency spikes will be present during the execution of the real-time enabled programs, therefore allowing for a precise and completely controlled execution.

3.3.1.1.5.1 Simulator Scheduler / Core

Simulator Scheduler / Core is the father process of the simulation execution. It is a program developed in C that takes care to handle all the process spawnings, the priorities of the processes, ports initialization and forwarding, time and synchronization.

Figure 55 shows the flow chart of the simulator scheduler. It reports the internal structure of activities, operations and the settings required to setup the processes in terms of numbers, types and other features presented hereafter.

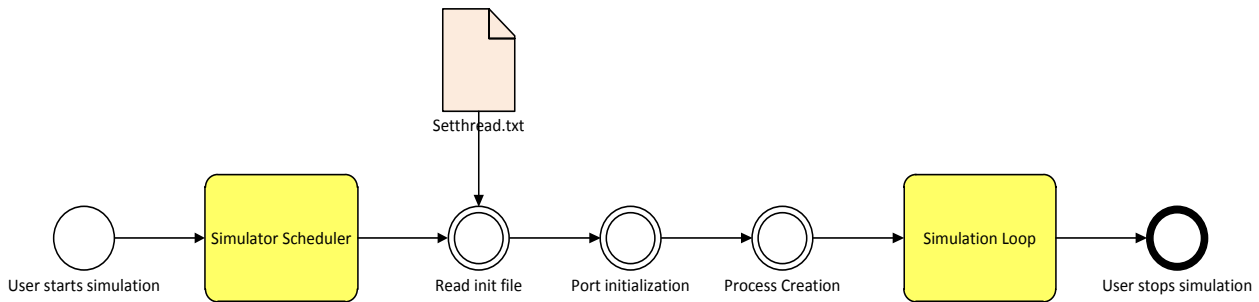


Figure 55: Simulator scheduler flow chart

3.3.1.1.5.2 Setthead.txt file

This file contains all the information needed by the scheduler to run a simulation. Going into details thanks to the Figure 56:

####

This line is used by the parser to confine the information regarding a process. The parsing continues until another line like this is found. When it is found, the Simulation Scheduler includes all the following information in another process.

1

Process number, it's used for identification purposes in the file.

Tempistica 1

Line used to inform the scheduler of which timing scheme must apply to the selected process. The scheme configuration has not been defined yet.

Priority 1

This line is used to impose a priority on the process in the OS. The number might range from 1 to 99.

Path

Path is the information on where the executable is stored in the file system. It will be used during the *execve()* call.

Pipe

These are lines to define the named pipes to be used.

- pipe: informs the parser that the line contains information on pipes
- pipe_imu: name of the pipe. Will be passed to the spawned processes for their uses.
- path: path to the named pipe
- mode: can be alternatively *r*, *w* or *rw*.

Serial

These lines define the serials to be used.

- serial: informs the parser that the line contains information on serials
- serial_pwm: name of the serial, to be used as a file descriptor when opening the port inside the process that uses it
- ttyUSB0: partial path to the serial, as initialized by the OS
- B115200: example of speed at which to initialize the serial

Setthread.txt parsing is done by successive calls to `fscanf` function, after doing the proper checks on what we expect to find with every call. All in all, the idea is that the structure of *setthread.txt* file is known and so we can hardcode the file parser behavior.

```

1 #####
2 1
3 tempistica 1
4 priority 1
5 path /home/lawrence/Dropbox/git/Simulatore/e-st@rII_test/Debug/e-st@rII_test
6 pipe pipe_imu /home/workspace/Simulatore/e-st@rII_test/ports/pipe_imu rw
7 serial serial_pwm ttyUSB0 B115200
8

```

Figure 56: Example of *setthread.txt* file

This file is read by the scheduler that “prepares” the conditions determined by the user setup.

3.3.1.1.5.2.1 Port initialization

Simulator core parses the information on ports included in the file *setthread.txt*, then format the information into the `argv` field, that will be used in the `execve()`, and passed to the children. Port opening is performed in the children programs, while the parsing is done in the core. This might seem less efficient than simply parsing port information on the child itself, but conceptually the reading of the configuration file is performed on the core, so the information on ports must be read in the same program.

3.3.1.1.5.3 Process creation

For the simulation initialization, all the information on how many process to spawn, which executable to launch in which process, and which ports is that process going to utilize are available.

```

182     if (pid == 0) {
183         int status;
184         char *envparam[] = {0};
185         set_priority(sched, policy, prop->priority);
186         status = execve(prop->identifier, args, envparam);
187         printf("Return not expected. Error in the execve (%d).\n", status);
188         exit(EXIT_FAILURE);

```

Figure 57: Code snippet on process creation

In the Figure 57, it is possible to notice some of the features explained in this chapter.

- Setting the priority is done in the child and before launching the executable with `execve`.
- Simulation loop then starts after the various `execve()` are called.
- The exit in case of failure report the signal number.

3.3.1.1.6 Python and Simulation configuration

Simulator has been configured by the core during the sequence described in the previous paragraphs. Simulation setup is instead performed through two Python programs called *Initializer* and *FileParser*.

3.3.1.1.6.1 Inizializer

The *initializer* manages the parsing of the database and the creation of the various *.c* and *.h* source files that will be compiled in order to generate the executable program that runs to perform the simulation.

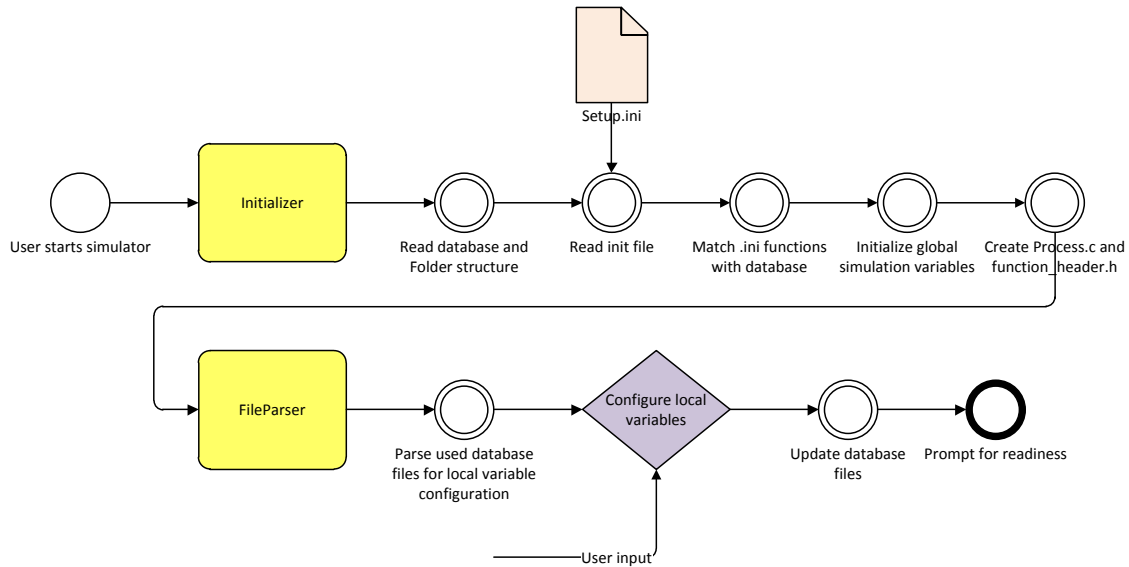


Figure 58: Simulator and simulation setup process

The parsing of the database is done dynamically: the *Initializer* instantiates the first root folder of the database, and then proceeds to inspect its content. For every object found, it checks if the name ends with *.c*, and adds it to the list, for later use. In case the filename doesn't end with *.c*, the *Initializer* assumes that as a folder (because, by design, only *.c* files or folders are stored in the database), and then calls the initial function again, this time considering the inner folder (here lies the recursion). The database is completely parsed by doing this proceeding for every object found in every folder. Object Oriented Programming (OOP) comes suitable in this situation because allows to assign more instances of the same object belonging to the folder

A few words must be spent on correctly parsing and checking the prototypes of the functions in the folders.

A typical C prototype looks like this:

```
void dynamics_RK4(*IN*/double B[3], double m[3], double *dt, double in0[3], double q0[4], double T[3], /*OUT*/double wib0[3])
```

A lot of information is stored in the prototype:

- the function name (*dynamics_RK4*)
- the return type of the function (*void*)
- the variables name, types and dimensions, and
- last but not least, the information on which variables are used as input in the function and which ones are outputs.

The *Initializer* must support any type of variable and any combination of input/output. Python is very versatile, and a custom algorithm has been developed to parse all the information and store them in a dictionary related to the function considered.

The checks performed on the consistency of the prototype range from checking that a variable with a certain name, dimension and type has not been defined differently in another function, to correctly formatting the prototype itself, and so on.

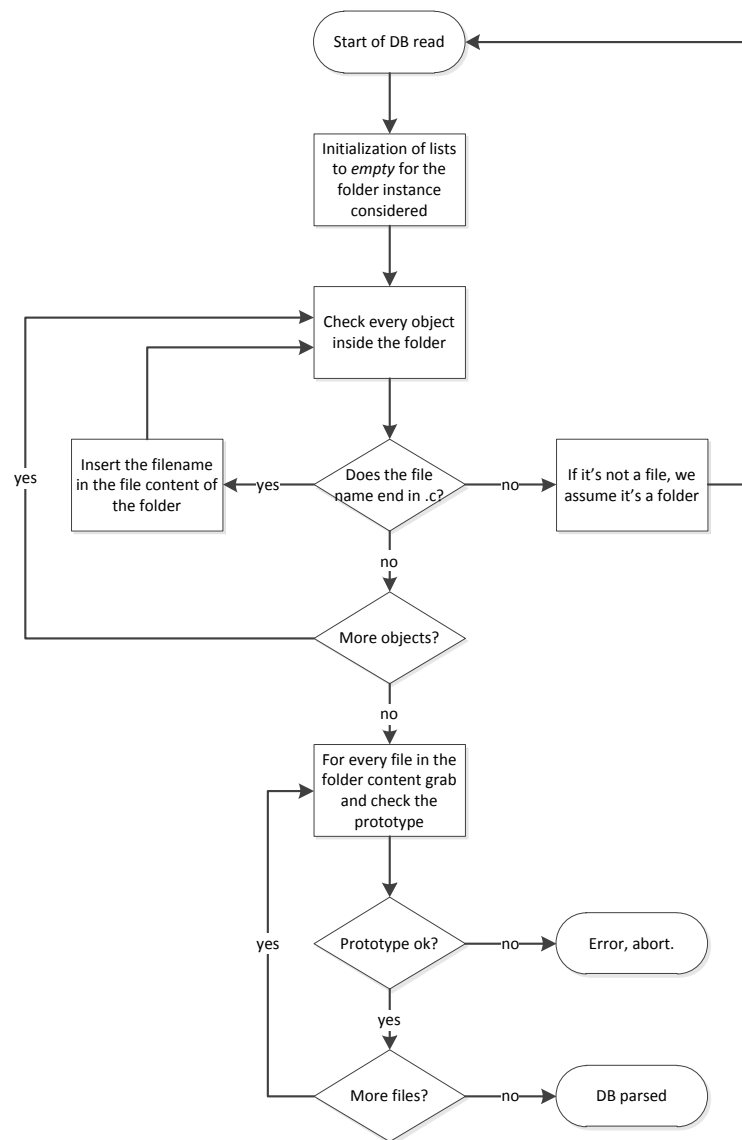


Figure 59: Iteration to parse the database

Figure 59 highlights in details how the models selection is managed: when the user builds the models flow, he/she creates, implicitly, a list of functions (one function corresponds to a model). When the list is completed, the program scan the choice made and associates for each selected element the corresponding C++ function, making the check of the prototype and verify that all the variables and constant values required for the execution of every function are properly defined.

3.3.1.1.6.2 Setup.ini

The *Setup.ini* file contains a list of functions that the user need picks from the database in the specified order: it correspond to the linear flow of the executable file.

Additional information is present in this file: the line '----' separates the functions called only once at the initialization stage of the execution from the functions that will be called in any iteration of the loop. During the process code creation, there will be checks on the function names contained in the *setup.ini* and the functions stored in the database, but this topic will be covered in the next section.

3.3.1.1.6.3 *Process.c and function_header.h*

The two files are created by the Initializer:

- *process.c* is the file with the *main* of the executable generated file
- *function_header.h* is a file containing all the prototypes of the functions used in the main.

The creation of the file *function_header.h* is pretty straightforward, because the prototypes directly derive from the *setup.ini* file. The creation of the file *process.c* is more complex: the structure of the file is defined, since there are interactions of this executable with others. Therefore the creation of the file (considering it a mere ‘parsing problem’) can proceed with a defined flow.

Initial lines contain:

- The included files
- Initialization of the ports, as defined by the input files
- Error occurred checking on port initialization

The following step performed is to grab every variable inserted in all the functions called in the stack (the *setup.ini* file), performing consistency checks among variables: in fact, the parsing stops if a variable is re-declared with different dimension or type.

Then the variable initialization should be initialized. The simulator asks an initialization value for every variable. If *enter* is pressed without any number, the initialization is completed putting default values.

For *int*, *float*, or *double* arrays, a prompt appears before the actual input request for initialization values: the user is asked whether the user wants or not to initialize the array to something different than zero; in case of a negative answer the array initialization defaults to zero. For *char* arrays, a length is inserted, to differentiate between strings (which obviously can be initialized to 0) and useful arrays (that can be initialized to something else).

After the variables initialization (and relative lines creation) the program parses the information on the *setup.ini* file, compares it with the stacked objects, and from each object’s prototype extracts the information to correctly call the function inside the program.

3.3.1.1.7 FileParser

Some model might have local constant or variables that should be initialized at the start of the simulation. FileParser is a program designed to take care of the initialization of the variables in a file that needs that need configuration.

The file is flagged as configurable by the initializer, during the parsing described above. The *is_configurable* flag is the string `/* CONFIGURABLE */` located in the first line of the file.

The configurable section is included inside the file between the flags `/*CONFIG_SECTION*/` and `/*END_CONFIG*/`, as shown in the Figure 60

```
9      /* CONFIG_SECTION */
10
11     double K_gi[3][3]={3.7764, 0.0153, 0.0762}, {-0.0070, 3.7228, -0.0973}, {-0.0001, 0.0624, 3.8955}};
12     double BIAS_g[3]={33000, 32729, 33215};
13
14     /* END_CONFIG */
```

Figure 60: Example of model setting parameters

All the variables contained inside the flags will be parsed, and a window will appear showing information on all the values found, that can be confirmed or set properly by the user. Pushing “enter”, the configuration is loaded and saved. The Figure 61 shows an example: the settings parameters of the IMU virtual model contained into the StarSim database.

When all these steps have been performed, the process is ready to be compiled, the executable program is directly generated.

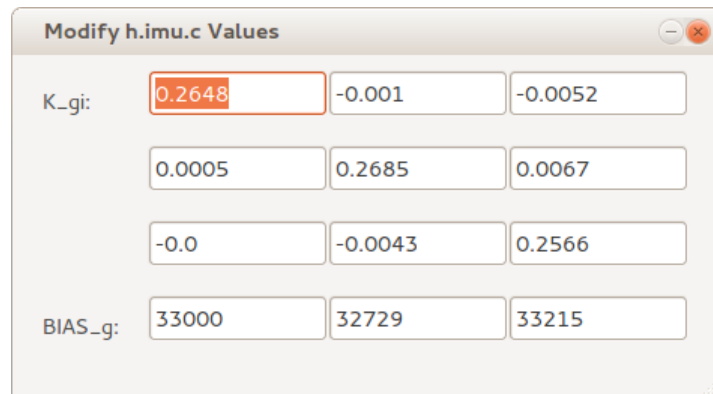


Figure 61: IMU settings window

3.3.1.1.8 Automatic Code Generation

The technology shown before is characterized by a high level of complexity on the code side but reduces the user interactions to *Yes/No* questions, to variable initialization (which can be skipped) made through selections within the GUI windows (see next paragraph). Remembering that skilled operators are required to perform the simulation, the simulator and the simulation initial configuration is simple and can be performed in a very straightforward way.

The process of automatically generate executable code reduces huge sources of bugs and problems in the creation of the simulation instances: the user does not need to handle the several functions of the simulator, including real time execution, priorities and time tagging, correct managing of all the processors that will execute the programs, and so on. This process is done automatically by the simulation unit, as described above, saving time and reducing boring activities. Moreover, it is easy to take under control the development because the code is reported on the user-friendly interface where it is also facilitated the navigation. The same generated code is user friendly in fact the user can directly see and control the code having a well defined structure and where variables are traceable and the execution flow is modular.

The “difficulties” is limited to build the simulation architecture as a LEGO construction where the objects are the various *.c* files that are included in the simulation initialization. Automatic code generation can handle local variable initialization, but for now no operation is done on the model itself, which has to be programmed by an engineer.

Moreover, StarSim provides the instrument to cross-compile the generated source code for two types of architectures but in future release should extend this number. The cross-compiling is supported for ARM scale architectures and MSP430 architectures and it is not included in v1.x release.

3.3.1.2 Interface unit

The interface unit has the main tasks to connect the simulation unit with internal and external modules and part of the entire simulator. Depending on the configuration chosen for the simulator, there will be communication between processes with different features. The interface unit is in charge to manage communications both in the case the two processes are executed by the same processor, and in the case the two or more processes are handled by two different controllers. In both cases, a data link with a precise path for the data exchange between the processes is essential.

There are multiple solutions to this problem, both in the case of a software link (for the case of two processes executed by the same processor) and in the case of hardware link (in the other cases).

StarSim has been designed to support a great number of communication types, in order to provide the user choose with the widest varieties of options when the simulation architecture is configured.

The interface unit is based on two big types of communication:

- Communication between processes executed by the same hardware or software interfaces;
- Communication between processes executed by different hardware or software interfaces.

3.3.1.2.1 Software communications

The communication between processes executed by the same hardware, three types can be individuated:

- File write/read
- Named and un-named pipes
- Unix Domain Sockets

3.3.1.2.1.1 File write/read

Data transfer between processes can happen by means of a file written by one process and read by another.

Common *C* functions *fprintf*, *fscanf* can be used in this case, while keeping in consideration two possible issues:

- A data formatting protocol needs to be known by both of the processes, in order to correctly parse the information
- Access races to the file might occur, since one process could try to open the file that is already opened by the other, this resulting in unspecified behavior.

3.3.1.2.1.2 Named and un-named pipes

The *pipes* are unidirectional data channels that can be used for inter-processes communication. They are implemented creating two file descriptors, ones used for reading and ones for writing.

Pipes can be created as *named* or *unnamed*. The unnamed pipes can be shared between two processes, one of which is created with *fork()*: this is because *fork()* shares the file descriptors among child. The drawback of unnamed pipes is that they cannot be delivered by a father to two children that then will use it for IPC (Inter-Process Communication) between the children: if the father creates the pipe, the two children are able to communicate between each other with the pipe. Unfortunately this is the case of the StarSim architecture in which the unnamed pipes are not taken into account.

StarSim uses named pipes shared simply by passing the path of the named pipe to the children on the computer file system.

The synopsis for the un-named pipes is the following:

```
#include <unistd.h>
int pipe (int pipefd[2]);
```

The array *pipefd* is used to return two file descriptors referring to the ends of the pipe: *pipefd[0]* refers to the *read* end of the pipe, while *pipefd[1]* refers to the write end of the pipe. Data written to the *write* at the end of the pipe is buffered by the kernel until it is read from the *read* end of the pipe on the other side. On success, zero is returned. On error, -1 is returned and *errno* is set appropriately.

The end of the pipe can then be read and written with the common *write()*

```
check = write(fd[1], "ciao", sizeof("ciao"));
if (check > 0) {
    printf("write riuscito\n");
    printf("check %d", check);
}
```

```

} else {
    printf("Error in write\n");
    return (EXIT_FAILURE);}

```

and `read()` functions:

```

check = read(fd[0], &buffer, 100);
if(check < 0) {
    printf("read ok\n");
    printf("%s\n", buffer);
} else {
    printf("Error in read\n");
    return (EXIT_FAILURE);}

```

For inter-process communication between two non-related processes, as in the case of SIL, named pipes were used.

Named pipes are just normal pipes, except for the fact that they are created by the program (or the user) and are physical files on the machine.

The command to create them from the terminal is

```
# mkfifo NAMED_PIPE
```

which has to be performed with super-user permissions: this, in addition, can be performed by the simulator performing a system call with the command

```
# system("mkfifo NAMED_PIPE")
```

In the code, pipes must be opened with the `open()` command, setting the rules for opening them (read only, write only, and so on).

```

pipe_pwm = open("FIFO_pwm", O_RDONLY);
pipe_imu = open("FIFO_imu", O_WRONLY);

```

Reading and writing to pipe is then simply performed utilizing the already introduced `write` and `read` functions.

3.3.1.2.1.3 UNIX Domain Sockets

In order to complete the frame of the possible software interfaces, UNIX Domain Sockets are described. Similar to named pipes, and typical of UNIX architectures, they are a data communication endpoint for exchanging data between processes executed by the same processor. Like named pipes, they do not need to share a common ancestor and API (Application Programming Interface) are present that simplify the interaction with this type of facility. Communication occurs completely within the operating system kernel.

3.3.1.2.2 Hardware interfaces

The communication between processes executed by different hardware can be guaranteed through different protocols and ports. At this moment, StarSim implements in C the following types of hardware interfaces

- Serial RS232: `serial_init()`
- USB: `usb_init()`
- LAN: `ethrnet_init()`

In the future release, StarSim can provide hardware communications also through CAN bus and I2C.

3.3.1.2.3 Data protocols management

StarSim gives the opportunity to manage and reproduce the coding and decoding operations on the data according to high level or custom data protocols. These operations result useful for SIL and essential for CIL and HIL simulations. User can define the format of the data intended as the header and closer characters of a raw string, and the sequence of the information and their cast (e.g. `char`, `int`, `float`, `double`). The implementation in C is made with structures as the follow example:

```

typedef struct str_out {
    char header[3];
    char com_obc;
    float q1;
    float q2;
    float q3;
    float q4;
    unsigned int gyro1;
    unsigned int gyro2;
    unsigned int gyro3;
    unsigned int acc1;
    unsigned int acc2;
    unsigned int acc3;
    unsigned int emf1;
    unsigned int emf2;
    unsigned int emf3;
    unsigned int pwm1;
    unsigned int pwm2;
    unsigned int pwm3;
    unsigned char flag_on;
    char closer[2];
} str_out;

```

(This is the protocol used for the communication of a self developed ADCS board and the OBC in the est@r program – so refer to Chapter 4 for details on the e-st@r program).

The operations of coding and decoding are performed by common UNIX functions mixed by self-written functions autonomously set by the simulation unit. More in details, for the coding the functions are:

```

format_packet(str_out, format-string)
write(number_serial, format_string, n)

```

For the decoding, the sequence of functions is:

```

read((number_serial, buffer, n);
validate_packet(buffer, data-vector);
extract(data_vector, str_in);

```

3.3.1.2.4 Interfaces Database

In the StarSim, the interface functions are gathered and organized in a database, as shown in the Table 26.

Categories		Name
Software Interfaces	File write/read	
	Pipes	Named
Hardware Interfaces	Serial	RS232
	USB	USB
	LAN	LAN
	I2C	I2C
	CAN	CAN
Protocol Management	Handling	Format_packet
	Handling	Read
	Handling	Write
	Handling	Generate_string
	Handling	Extract

Table 26: Interfaces functions database

3.3.1.3 *Model database categories*

The importance of the models within the proposed methodology is unquestionable. The paragraph 1.4.4.1 explains the philosophy adopted in the StarSim in reference to the models, virtual and physical, their features and organization.

The fundamental part of any simulation is the ability of modeling the elements of the system under study. A database with different mathematical and stochastic models, divided in several categories, has been built and is periodically updated. The simulator has been designed according to the principle that any simulated system should behave as the real ones not only in terms of numerical or physical outputs, but also in terms of interface with the other elements in the simulation loop. Potential anomalies in actual sensor's measurements shall also be reproduced in the virtual sensor. The level of models' detail shall be carefully evaluated taking into account the accuracy of the simulation on the one side, and the computational cost on the other side. Usually, increasingly more detailed models are used according to the advancement of the design. The models come from literature when available or they are derived from information given by the manufacturers, or they can be obtained from data acquired during open-loop testing of the specific equipment.

3.3.1.3.1 The spacecraft equipment/components models

The Simulation Unit includes electrical emulation of equipment: e.g. the output value of each sensor is computed by the simulator and it is read by the embedded system under test. In the same way, the embedded system commands the actuators by means of proper control signals and it sends back to the simulator the control values.

More in general, every simulated system should behave as the real one not only for the numerical or physical outputs but also for the type of interfaces with the other tested elements. Furthermore, any potential anomaly in actual sensor's measurements should be reproduced in the corresponding modeled sensor.

StarSim allows loading data and functionalities of the simulated equipment in order to configure the simulation parameters during initialization. Examples of settings are:

- Characterization of the modeled equipment
- Default values
- Definition of simulation parameters like step time, integration method, sequence of models loading
- Configuration of the telemetry and log parameters
- Configuration of the protocols and the kind of interfaces (for HIL)
- Calibration parameters

In the high fidelity models, components are modeled in such detail that it is possible to simulate nominal operating modes as well as failure modes and not idealities are taken into account. The failure functionalities inside the equipment models allow a consistent reproduction of failure symptoms of the real spacecraft components for tests of the on-board software. In addition a functional representation of the spacecraft on-board harness can be included in the simulation comprising power supply harness, signal harness reflecting analog, digital and data bus connections between onboard equipment.

3.3.1.3.2 The space mission models

The space mission models mainly refers to space environment models that cover a great importance both for the verification and validation issues and for drastically reduce the costs and the resources involved in a test campaign. Validation passes, as explained in paragraph 1.2, through the verification in the real conditions. They are often too difficult or impossible to reproduce exactly and all together. This implies that environmental condition and dynamics effects on space system are conveniently reproduced through virtual models or, in some cases, hybrid models.

3.3.1.3.3 The database

StarSim models are organized within a database, according to the following main categories:

- Devices and Equipment,
- Mission: Environmental and S/C motion
- GSE
- Special Functions
- Transformation and conversion
- Display and Visualization

The following tables highlight all the models and functions contained in the databases, already divided in categories.

3.3.1.3.3.1 Devices & equipment

This paragraph contains the list of devices and equipment model that should be present into the related database. [9], [10], [11].

Categories		Name	Inputs	Outputs
Sensors	Attitude determination	Magnetometer	modeled EMF (Bb)	measured EMF (Bmis)
Sensors	Attitude determination	Gyroscope	Angular rate (wib)	measured angular rate (wib_mis)
Sensors	Attitude determination	Sun Sensor	satellite attitude (q)	Normalized Sun vector in the sat body frame (S)
Sensors	Attitude determination	Star Sensor	Images of stars taken by sensors	Difference angle between measured and stored star unit vectors (qstar)
Sensors	Attitude determination	Horizon sensor	Shape of Earth's limb (Earth's IR radiation) - (EarthShape)	Direction of Earth (Nadir)
Sensors	Orbit determination	GPS/Galileo		
Sensors	Orbit determination	Accelerometer	real acceleration (acc)	measured acceleration (a_mis)
Sensors	Thermal	NTC&PTC	Temperature (Temp)	measured temperature (Temp_mis)
Sensors	Navigation Sensor	Encoder	Angular rate (w_enc)	Position (x,y,z) in the body frame
Signals Handling Devices	Converters	ACD	analog signal (in)	digital signal (out)
Signals Handling Devices	Converters	DAC	digital signal (in)	analog signal (out)
Signals Handling Devs	Logic	PWM	command (command)	duty cycle (DC)
Actuators	Attitude Control Actuators	MagneticTorquers	real dipole moment (or MT) current and EMF (m)	dipole moment (mtorquer)
Actuators	Attitude Control Actuators	VernierThruster	command torque (Treal)	thruster torque (Tth)
Actuators	Attitude Control Actuators	PermanentMagnets	generated dipole moment (mpm)	generated torque (Tpm)

Actuators	Attitude Control Actuators	ReactionWheel	Voltage (V), Current (i)	motor angular velocity (w_{rw}), generated torque (Trw)
Actuators	Orbit Control Actuators & Propulsors	Thruster	comamnd torque (T)	thruster torque (Tthruster)
Power Sources		SolarCell	attitude (q)	Voltage (Vsp), current (isp)

Table 27: Models database of devices and equipment

3.3.1.3.3.2 Mission - Environment & S/C motion

Table 28 lists the S/C motion and the environmental models [11],[14] and alii.

Categories		Name	Inputs	Outputs
S/C motion	Linear relative motion	Hill's equations	sum of forces (sumF)	relative position and relative velocity (xr,yr,zr)
S/C motion	One body orbit propagation around the Earth	SG4 propagator equations	sum of the forces (sumF)	orbit position (x,y,z) - (LAT,LON,H) and velocity (vx,vy,vz)
S/C motion	One orbital body Dynamics	Euler/Newton equations	sum of the torques (sumT)	body angular velocities wrt inertial frame (wib)
S/C motion	One orbital body Kinematics	Euler/Newton equations	body angular velocities wrt inertial frame (wib)	attitude (q)and body angular velocities wrt orbital frame (wob)
Environment	Earth Magnetic Field	IGRF	Attitude(q), time (tt),	EMF (B)
Environment	Thermal	Heat fluxes calculation	attitude(q)	tempeture (Temp)
Environment	Radiations	BitFlip injection	right value (input_name)	failed values (fail_value)
Environment	Radiations	Latch-up Injection	right value (input_name)	
Environment	Attitude perturbations	Aerodynamics Torques	altitude (H), atmospheric density (rho_atm)	aerodynamic torque (Ta)
Environment	Attitude perturbations	Gravity Gradient Torques	orbital position of the spacecraft (LAT,LONG,H), spacecraft attitude	gravity gradient torque (Tgg)
Environment	Attitude perturbations	Magnetic Torques	Orbital position (LAT,LONG,H), EMF(Bb), Magnetic dipole(m)	magnetic torque (Tm)
Environment	Attitude perturbations	Solar Pressure Torques	Attitude of the spacecraft (q),(equivalent area exposed to the Sun), sun position (S), arm (cp-cm)	solar pressure torque (Tsp)
Environment	Orbit perturbations	Atmospheric Drag	altitude (H), atmospheric density (rho_atm)	atmospheric drag (Fa)
Environment	Orbit perturbations	Solar Pressure	Attitude of the spacecraft (y)(equivalent area exposed to the Sun), sun position (S)	Total force acting on the spacecraft

Table 28: Models database of environment and S/C motion

3.3.1.3.4 GNC strategies

One section of the database is devoted to the GNC strategies and related models (see Table 29). [13], [14], [15], [16].

Categories		Name	Inputs	Outputs
Control strategies	2D motion control strategies	Way Point	wp positions in the body frame (WP)	-
Control strategies	Attitude/trajctory control laws	Fuzzy	state variables (x)	commands (y)
Control strategies	Attitude/trajctory control laws	Hinfinity	state variables (x)	commands (y)
Control strategies	Attitude/trajctory control laws	LQR	state variables (x)	commands (y)
Control strategies	Attitude/trajctory control laws	Neural Network	state variables (x)	commands (y)
Control strategies	Attitude/trajctory control laws	PID	state variables (x)	commands (y)
Determination strategies	Deterministic & Statistic methods	ESQ		
Determination strategies	Deterministic & Statistic methods	QEST	model values (in_model) and measured values (in_mis)	estimated attitude (qest)
Determination strategies	Deterministic & Statistic methods	Qmethod	model values (in_model) and measured values (in_mis)	estimated attitude (q-method)
Determination strategies	Deterministic & Statistic methods	TRIAD	model value (in1_mod) and measured value input1 (in1_mis), model value (in2_mod) and measured value input2 (in2_mis)	estimated attitude (q_triad)
Determination strategies	Deterministic & Statistic methods	KF	plant state variables (x) and measurements (mis)	estimated attitude (qhat)
Determination strategies	Deterministic & Statistic methods	EKF	plant state variables (x) and measurements (mis)	estimated attitude (qhat)
Determination strategies	Deterministic & Statistic methods	UKF	plant state variables (x) and measurements (mis)	estimated attitude (qhat)
	Orbit determination	propagation from TLE	NORAD	lat,long,altitude
Guidance strategies	Pointing		desired values (xd)	
Guidance strategies	StationKeeping		desired values (xd)	
Time management		SimTime Management		sim time
Time management		UTC time management		year, month, day, hour, minute, second

Table 29: Models database of GNC strategies and special functions

3.3.1.3.4.1 Transformations and conversion

Categories		Name	Inputs	Outputs
Conversions	Angular	euler2quat	euler angle ($\varphi\theta\psi$)	quaternion (q)
Conversions	Angular	quat2euler	quaternion (q)	euler angle ($\varphi\theta\psi$)
Conversions	Angular	rad2degrees	radiant	degree
Conversions	Angular	degrees2rad	degree	radiant
Conversions	Length	meters2feet	meter	feet
Conversions	Length	feet2meters	feet	meter

Conversions	Rotation Matrix	rotation inertial2orbital	attitude (q), vect_in	vect_out
Conversions	Rotation Matrix	rotation orbital2body	attitude (q), vect_in	vect_out
Conversions	Rotation Matrix	general rotation	phi,theta,psi, vect_in	Vect_
Math operations		3x3 cross product	vect1_in, vect2_in	vect_out
Math operations		matrix determinant	matrix	determinant
Math operations		matrix eigenvalues & eigenvector	matrix	eigenvalues and eigenvectors

Table 30: transformation and conversion functions database

3.3.1.3.5 Ground Support Equipment

Categories		Name	Inputs	Outputs
Rotational	1D platforms	Gimbal platform	DC for motors (DCmotors)	platform motor voltage and current (V, I)
6DoF	Robotic arms	CADET robotic ARM		
Solar radiation	Power suppliers	ISO-32	Vset,Iset	V_alim, I_alim

Table 31: Models database of GSE

3.3.2 Ground Support Equipment

The GSE are all those elements (instrumentations, devices, systems, assemblies, cables) that

- give/receive stimuli to/from *test object* according to the values computed by the simulator;
- serve to perform the simulation but they belong neither to the simulation unit nor to the test object

Three main categories of GSE can be individuated:

- GSE for stimulation of the test object
- GSE for externally power the test object and other parts involved in the simulation session
- GSE for the communication/ exchanging of data between two of the main “actors”, i.e. GS and test object, test object and simulation unit, the Control Console and the Simulation Unit. Pay attention that GSE for communications are to be intended as the equipment (cables, connectors, radio- module, and filters) that allow the physical connection.

3.3.2.1 GSE for sensors stimulation

Hybrid test benches enable two main types of “Hardware in the Loop” verifications which include more equipment than the embedded smart board (i.e. processors, controllers: the *Open Loop* test and the *Closed Loop* test.

3.3.2.1.1 Open Loop test

“Open loop” tests foresee that real spacecraft sensors, actuators or payload are connected to the on-board computer via harness.

Initial tests goal should evaluate if:

- the equipment can be controlled correctly and,
- the on-board computer receives all acquisition data and status telemetry without errors from the equipment.
- the equipment occurrences are connected correctly to the on-board computer ports.

For a subset of critical sensors it might be necessary to stimulate them dynamically and quantitatively, to measure whether the entire system control loop with OBC and sensor in the loop operates as

desired. Depending on the different sensor types, stimulation equipment is required for each. An example derives from a sun sensor: it could be stimulated with a lamp, able to reproduce a radiation compatible with Sun radiation. The sensor position can be controlled dynamically during test from the control console via an electrically commendable two axis gimbals.

Other example applied to StarSim is an IMU characterization (the Sparkfun Atomic 6DoF).

The test bench has been built in the StarLab at Politecnico di Torino (with Eng. Raffaele Mozzillo and Mr. Davide Falsetti) to properly test the behavior of an IMU; it consists of:

- an electric motor EMG30 (used with and without gearbox);
- facilities supporting cables necessary to power the motor, the platform and of course for serial connection to PC;
- a RS232 adapter with the task of converting the voltage at the desired levels;
- a rigid surface in plexiglass for the installation of the platform.

The final result is shown in Figure 62, where it is possible to see more details.

Through the simulator has been possible to control the motors speed, performing an “open loop” test that allows characterizing the IMU output. Knowing that the measures provided (this particular IMU provides measurements of angular velocities as voltages) requires a proper conversion from digital to analog measure of voltage using the following relationship:

$$V_d = \frac{V_a \text{res}}{V_{ref}}$$

(where V_a is the analog voltage measure, V_d is the digital voltage measure (provided by IMU), res is the resolution ($res = 210 = 1024$) and $V_{ref} = 3.3[V]$ is the reference voltage), the simulation has been built and setup. It foresees:

- the setup of the serial interface
- the definition and calibration of the sensors parameters
- the setup of the GSE (the motors and their drivers)
- the definition of the commands and the operation modes for the motors
- the acquisition of data from sensor
- the definition of the saved files and the “sensible data” that will be stored in

Figure 62 shows the results obtained from the simulation session.

3.3.2.1.2 Closed Loop test

“Closed-loop simulation”: the simulated system can be used to calculate attitude and position of the spacecraft. With this information, the Sun position detected by the sensor can be provided by the simulator and the corresponding control can be applied as well as the angular velocity measured by gyros. The processor then receives the measured signals from the real sun sensor and gyros as if it were in Earth orbit. The software performs the calculations and returns the attitude control cycle. The stimulations and/or measurement infrastructures for such closed loop tests can completely differentiate between the system types under test. Their functionalities are individually determined by the definition of tests which are to be run on the installation. The same kind of procedure can be thought for actuators tests.

Details about this kind of configuration are contained in the Chapter 4, where the HIL simulations for e-st@r CubeSats are deeply explained.

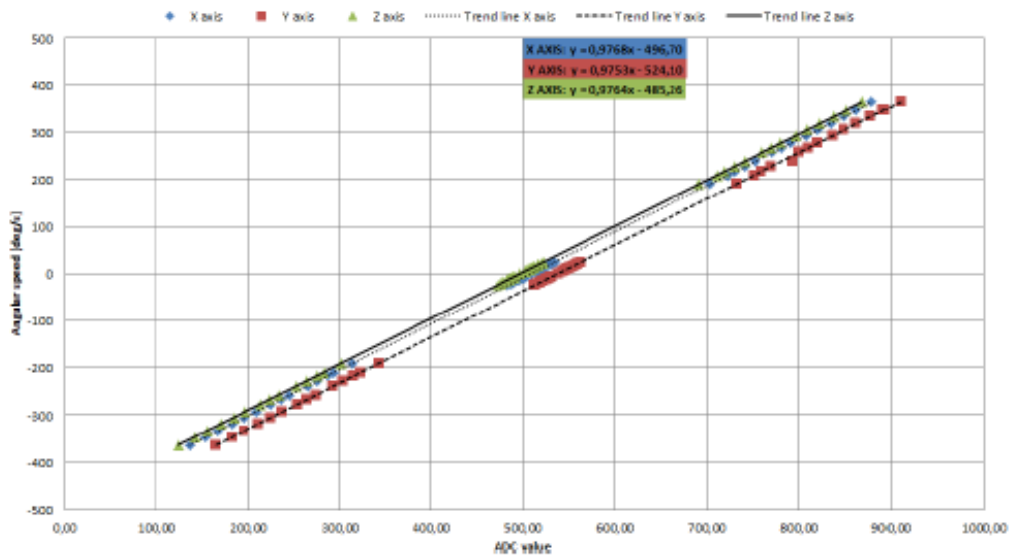


Figure 62: Plot of the outputs for IMU Sparkfun characterization

3.3.2.2 GSE for power

A typical GSE in the test benches is a power supply system. To differentiate from standard laboratory power supplies, the test bench supply usually is called the "Power- Frontend" and it supplies all units of the test bench, if the EPS of the space vehicle is not involved in the running simulation session. Depending on the project complexity such a Power-Frontend is either controlled manually or can be commanded by simulation unit. Clearly, over/under voltage/current circuits provide the right protection via latch-up limiters.

For the StarSim, the Power GSE is mainly represented by:

- cables and current and voltages transformer used to connect the great number of the CC devices (Laptop and PC) and boards (development and embedded board and physical models of sensors and actuators) with the C.A. 220V @ 50 Hz,
- power supply units which take the place of the batteries, solar cells and other electrical generators and provide power directly to the test object. In some configurations, they are manually setting (as made traditionally) or (more interesting) they are commanded by the Simulation Unit. The main example for StarSim is given by the ISO-TECH IPS 3202 power supplier. It has three outputs commendable via RS232 formatting the commands with the data protocol (available from the manual).

3.3.2.3 GSE for telecommunication and data exchange

TM/TC devices are GSE for telecommunication and data exchange. It is possible to individuate two subcategories: *Simulator Frontend* provides the communication between the test object and the simulation unit, *TM/TC Frontend* provides the communication between the test object and the Ground System.

3.3.2.3.1 Simulator Front-end

Simulator Frontend device serves to connect the on board hardware with the still simulated rest of the system. It can consist of a set of interface boards, transferring signals from the real processor/controller to the simulator respectively to return simulated data to the controller. Since in such a HIL configuration the simulator has to respond in real time and thus has to run on a real-time operating system, also the data bus system for interface boards has also to provide real time data transfer features. Particular attention shall be paid when the interface boards, the test cables and the hardware interface on the controller side are connected because they must be electrically compatible,

requiring an accurate design of electrical input/output characteristics down to corresponding signal compatibility measurements. Another function (that Simulator Frontend shall guarantee) is the synchronization between test object and simulator, avoiding numerical problem and mostly transmission error in data protocols.

3.3.2.3.2 TM/TC Front-end

TM/TC devices allow the telecommunications between the test object and the ground system. They are necessary because telecommands (formatted following specific protocol) are sent to the test object (through the communication system of the spacecraft) in a real flight mission from the Ground Station. The on-board computer (in a “star architecture”) or the specific processor/controller of the subsystem (in a “distributed architecture”) receives the packet e handle the information. The way back from the onboard computer/subsystem’s processor is similar. Packets with data telemetry are formatted and transferred to the on board communication system, which sends to the ground. The Ground System receives the signal, extract and visualizes the information.

An example of TM/TC Front-end is the Ground Station equipment. Figure 63 shows the blocks-scheme of the GS developed for the e-st@r program and used for any kind of simulations during the test campaign. It can be considered as part of the StarSim, completing the global architecture

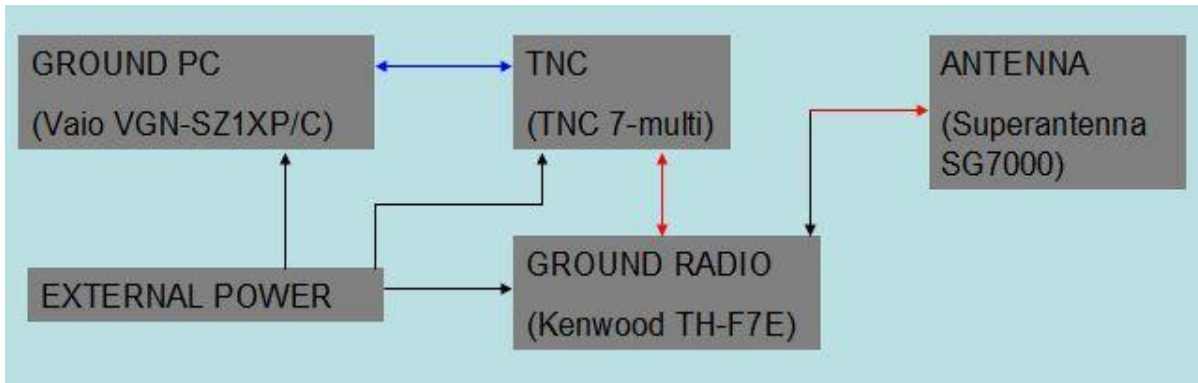


Figure 63: Mobile Ground Control Station Scheme

The GS is mainly devoted to receive the downlink data from the satellite so its configuration is done in this perspective. It is constituted by a dedicated Notebook (named NB-MGCS) and a Kenwood TH-F7E radio (Figure 65) with its antenna. For the short range communications the antenna of the radio is sufficient, for next tests at medium and long distance the antenna will be replaced by an appropriate antenna (Lafayette SG7000) (Figure 64).

The TNC (Figure 66) connects the terminal to a radio transceiver. Data from the terminal is formatted into AX.25 packets and modulated into audio signals for transmission by the radio. Received signals are demodulated, the data unformatted, and the outputs sent to the terminal for display.

In addition to these functions, the TNC manages the radio channel according to guidelines in the AX.25 specifications. The interface is typically constituted by analogical audio signal and a line for PTT and squelch. The connector that is normally used for all TNC is the Mini DIN 6 pin (actually classified in the ISO-9002 to standardize transceivers data ports). The TNC-7 multi is shown in figure 20 below:

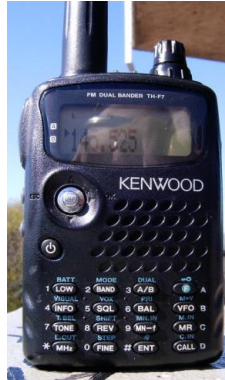


Figure 65: Kenwood TH-F7E



Figure 66: TNC-7 multi

Figure 64: LaFayette SG7000 antenna

Suitable software is needed in order to receive the data encoded with the chosen protocol (i.e. the KISS AX.25 protocol for e-st@r program), as sound input, and decode its contents. The choice fell on free software: AGW-PE. This software, with the relative driver, allows to displays the received packet on video. The Kenwood TH-F7E radio will receive the downlink data and transmit them as audio input to the microphone input of the NB-MGCS. At the same way, this software allows to set destination and source address and write the command message by the keyboard packing them into the protocol and transferring it to the radio using the PC earphones output.

The interface between PC and radio is guaranteed by special cable that is built expressly for this application. This cable allows the communication both for downlink and for uplink: in fact, an audio double stereo jack cable connects the PC earphones output with the radio microphone input; in the same way, an audio double stereo jack cable connects the PC microphone input with Radio the earphones output.

Moreover, there is a third connection between the radio Push To Talk (PTT) line and the PC USB port in order to enable directly and automatically the transmission of the uplink packet avoiding each time to push the Radio PTT button.

For StarSim, generic and more specific GSE for communication and exchange data have been designed and built. Some of them derive from previous concluded program or are bought as plug&play devices. In Figure 67 and Figure 68 the blocks schemes of a StarSim Frontend are reported. It contains different kind of hardware communications interfaces together with a Power GSE that regulates and distribute power at different CC voltages starting from the CA 220V@50Hz source.

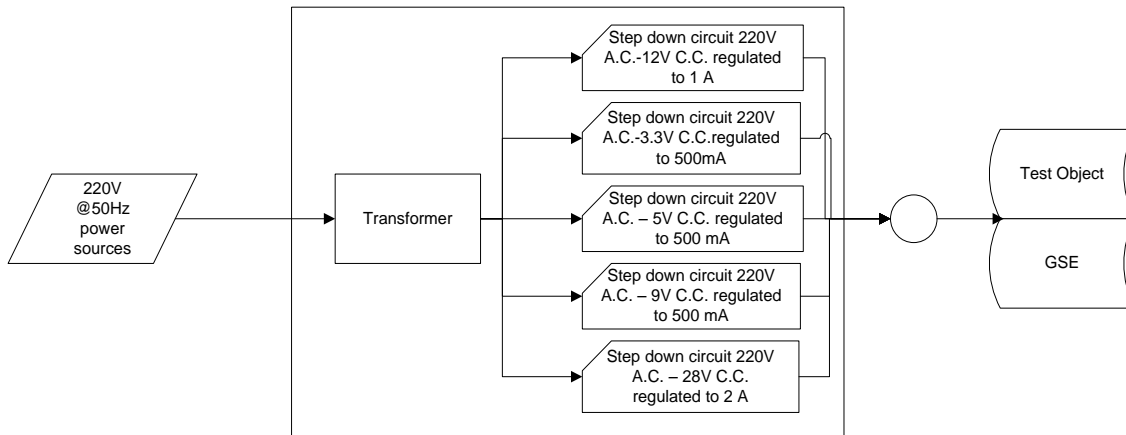


Figure 67: Blocks scheme of the power frontend for StarSim

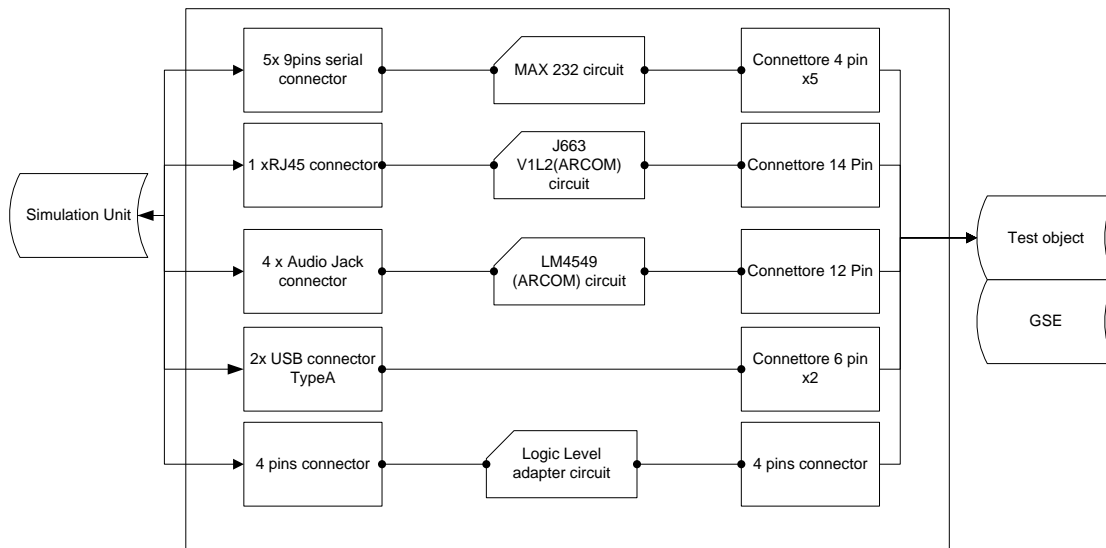


Figure 68: Blocks scheme of the TM/TC Front-End for StarSim

3.3.3 Control Console

The Control Console (CC) puts in communication the Simulation Unit with the user, enabling the control of the simulation session in any moment.

- Before the simulation execution, it favors the definition, the configuration and the setup of the architecture and parameters.
- During the simulation execution, it permits the real time monitoring and control as well as the live visualization of values and trends of the sensible parameters and variables.
- At the end of the simulation execution, it allows the data handling and the requirements verification

From the hardware point of view, the Control Console is a dedicated computer or a network of computers: Their interfaces are:

- LAN or Ethernet cables to be connected to the World Wide Web;
- RF or Wireless devices at different frequency with respect to the frequencies used by the test object, or wired through the debug ports located on the front-ends;
- Plug and Play cables.

In the StarSim v1.0 the Control Console stays in the WS with the Simulation Unit.

3.3.3.1 GUI (*Graphical User Interface*)

The GUI is the element that effectively permits the interaction man-machine. In fact, the GUI is a software program devoted to help the user in any phase of the simulation through a graphical organization that simplify the interaction between the user and the management of parameters and results management. An intuitive approach simplifies the user activities using graphic elements like virtual buttons, boxes, plots and tables. For details about the GUI of StarSim see the paragraph 3.4.

3.3.4 The test object

The test object is the system (or some of its parts) that would be investigated during the verification campaign. It can be constituted by:

- Embedded Systems: electrical board based on a “smart unit” (micro-processor/controller, PC104, FPGA, ASIC), circuits and devices needed for the *test object* good working, i.e. timers and synchronization systems, volatile and non volatile memories and input/output logical and physical ports;
- equipment includes mechanical, electrical, and RF devices. For the GNC, they are actuators, sensors, drivers logic circuit, and signal conditional board; other examples are engines, radio-modules, antennas, solar panels, batteries and fuel cells, etc...

Moreover, also the algorithms and software can be considered as real objects of the test.

3.4 How to use the simulator

This paragraph provides an overview of the activities that a user shall or could make with StarSim. In particular, the instructions and steps to build a simulation architecture, select and setup models and interfaces, execute the simulation and visualize and handle the results are briefly presented.

3.4.1 Simulator main window

The definition of the simulator architecture is a simple step to perform when the user has already designed the architecture of the simulation test he wants to be executed. This condition is valid for the StarSim v1.0, successive versions shall foresee design tools based on SE rules able to provide design output such as block schemes or functions/equipment matrices from which the software of the StarSim self-generating the simulation architecture.

After the launch of the executable file *StarSim*, the main window appears and it has six buttons:

- Select the simulator architecture
- Select the models
- Select the simulation parameters
- Select the outputs
- Run
- Data repository

Figure 69 shows this main window.

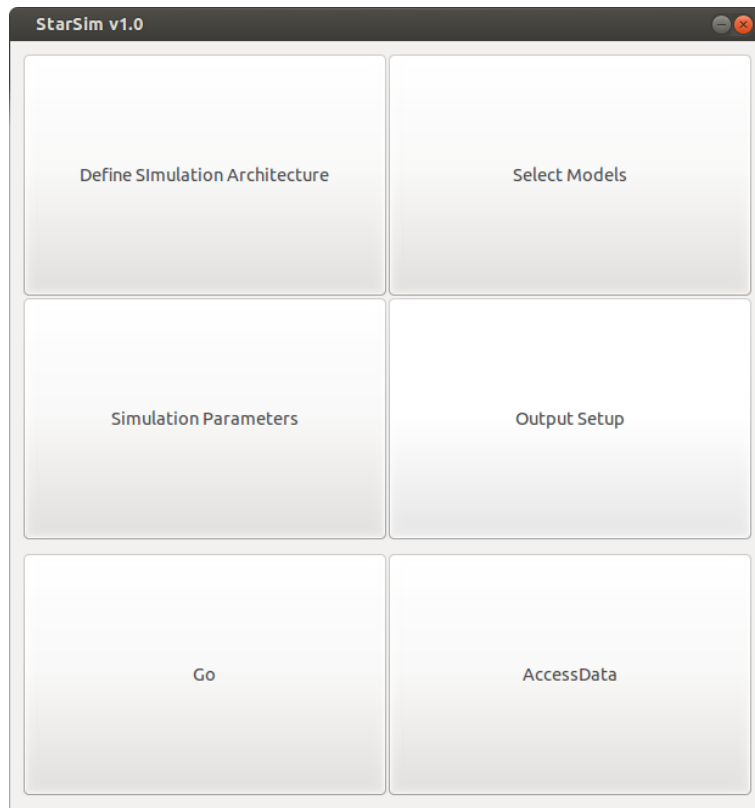


Figure 69: StarSim main window

3.4.2 Simulator architecture configuration

The following choices shall be made in order to have the desired configuration in relation with the type of simulation (AIL, SIL, CIL, HIL, hybrid). In this phase the user shall take decision on:

- Number of processes to create;
- Number of processors available for the simulation;
- Number and type of software communication ports;
- Number and type of hardware communication ports.

Figure 70 shows the first window of the GUI devoted to the definition of the simulator architecture.

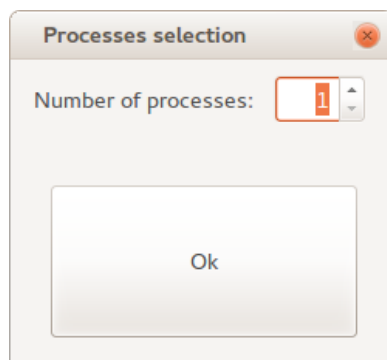


Figure 70: Initial process section window

Thickin the number in the box and pushing *OK*, the number of processes are setup. According to this number, the simulator will propose windows (such as that in Figure 71) in order to define the properties of each process. The priority, the path with the future name of the generated code and the number of interfaces shall be setup by the user. The number of interfaces carries out the selection of their type (hardware or software): the user can specify the type, the name, the path, the speed or other features of the interface through the labels automatically generated according to the set number .At the end of any process setup the user push button *SAVE* in order to save the configuration re-loaded in future and the button *OK* to fix the current solution.

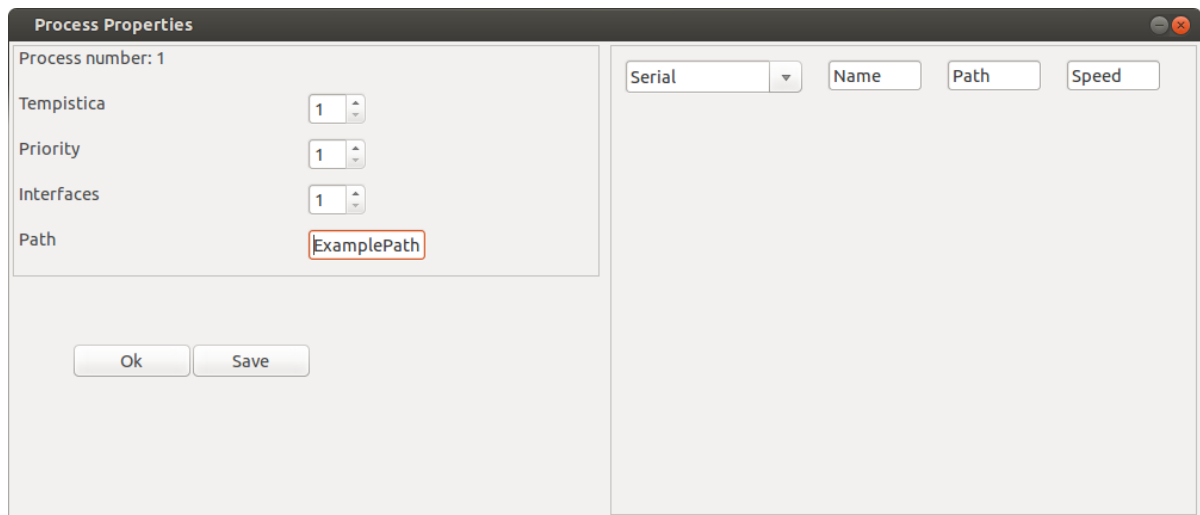


Figure 71: Process selection, main window

At the end of all the selections, the *setsimarch.txt* file is generated in an automatic way (see paragraph 3.3.1.1.5.2)

The *setsimarch.txt* file contains the following information for any process:

- First line, four '#': #####
- Second line, process number (starts from 1): 1
- Third line, timings requirements (starts from 1): tempistica 1
- Fourth line, priority setting (ranges from 1 to 99): priority 1
- Fifth line, 'path' followed by the path to the executable to be launched. In case the executable lies on another processor input 'crosscompiled': path /home/ws1/Repository/ExampleFolder/exampleExacutable
- Sixth and following lines contain the information on how many interfaces are needed and which type and settings they have.

3.4.2.1 Define the interfaces

As stated before, sixth and following lines in *setsimarch.txt* reports the information on the hardware/software interface. These lines contain:

- the software interface features: in *pipe* case, the label 'pipe' followed by name of the pipe, path to the named pipe in consideration, read/write mode expressed as 'r', 'w', 'rw' are the required setting: e.g. *pipe pipe_imu /home/ws1/Ports/Pipes/examplePipeName rw*
- the hardware interface parameters: for example a serial port, it shall be configured with the label 'serial' followed by *serial name*, the address (usually in the form of 'ttyUSB0', 'ttyUSB1', and so on, in case of serial-USB converters are used, or 'ttyS0', 'ttyS1', and so on in case of pure serial ports). Moreover, the baud rate, the parity check and control flow should

be defined. If no settings are defined the default values are setup, e.g. *serial serial_pwm ttyUSB0 B115200*. Similar examples can be done for USB ports, Ethernet/TCP-IP, GPIO, and others.

All the settings can be configured through the dedicated GUI windows (Figure 71).

3.4.3 Define the models flow

Models flow definition reflects the idea that the user needs to define which are the actual models involved in the simulation, after having defined the architecture of the simulator.

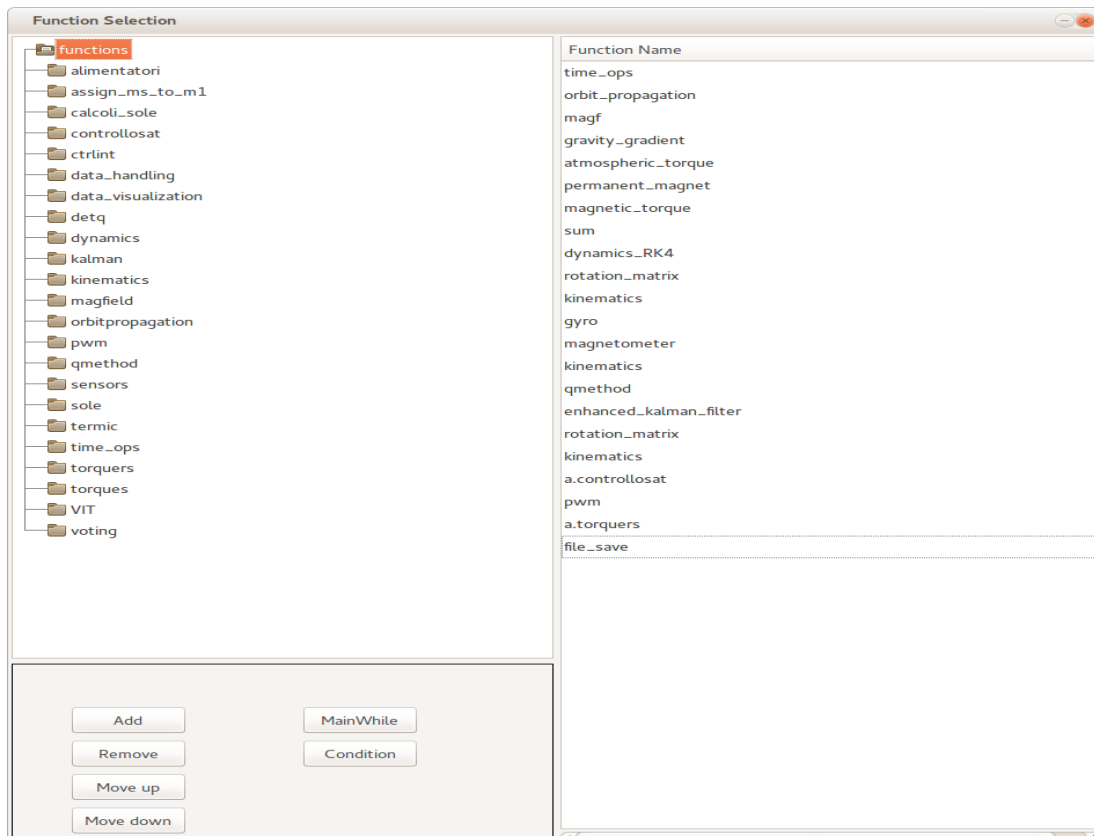


Figure 72: Models selection window

To choose the models and specify their order, the user simply interacts with the relative GUI window. Here, he can find a database explorer and select and drag into the near window the chosen models in the execution list configuration. In addition, the order can be directly modified dragging the functions in the desired place. Once the order has been defined it is possible to save it the files *models_flow.txt* in order to make it available for future use.

The Figure 72 shows “MODELS SELECTION” window where the database of models is in the left side and the source code generating by the choices is in the right side. The user can select the models exploring the database and push the button *ADD* to add the models in the flow. A model can also be removed with the button *REMOVE*, moved up and down in the flow sequence with relative buttons. Finally, the selected models are directly put on the loop but through the button *MAINWHILE*, he can leave it out of the loop: this is useful for those functions, e.g. *compare* devoted to post-processing activities or *gettimeofday* that setup the UTC before the starting of the running. The button *CONDITION* allows the setup of particular conditions that change the execution flow in runtime, i.e. it is used to define a mission profile.

In the future release of StarSim, Automatic models definition can be performed if the whole process of mission and system design has been completed utilizing specific software (SysML, UML) or future StarSim libraries. In this case an “allocation matrix” shall be available and the order of execution will be automatically created by parsing on that table.

3.4.3.1 Model configuration

Both in the case of automatic and manual models definition, the setup operations on models shall be made because a model can contain local variables and constants that need to be initialized or defined, and the success and correctness of the simulation is directly related to the information stored in these variables.

A dedicated GUI window will pop-up (see Figure 61) in the case the inserted model contains parameters that need to be initialized by the user. For any model, default parameters values are already loaded and are used if user does not proceed to define them accordingly. For every process selected will exist an own models flow characterized by the sequence of the chosen models. Behind each model there is a C++ function. At the end of the models flow definition a C++ code skeleton is generated and it is ready to be compiled and cross-compiled.

3.4.4 Simulation Setup.

3.4.4.1 Define the simulation parameters

Once all the configuration of the simulator has been performed in terms of simulation architecture, models flow and interfaces selection, the user can proceed to set the simulation parameters. These parameters are:

- *start time* and *stop time*: **tstart, tstop**
- *integration time* and *simulation step time* **dt, tt**
- *real time/no real time execution*: **realtime_flag**
- chosen orbit through the orbital parameters, **raan,e,I,omega,M,N**

Other parameters are normally loaded for any simulation: they are universal constants such as **Gk=6.67e-11**, **g0=9.81 m/s²**, **REarth=6378137 e6 m**, **MEarth=5.9742e24 Kg**.

A dedicated window in the GUI windows (Figure 73) allows these settings and, if required by the user, previous stored parameters definitions can be loaded from the data repository: in fact, all the configurations will be stored in a file *sim_parameters.txt*.

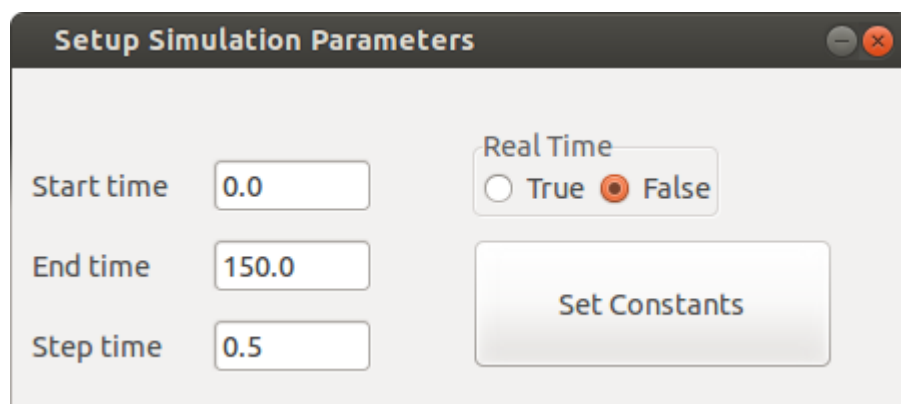


Figure 73: simulation parameters window

3.4.5 Define the output files

The simulator allows saving the sensible information related to the simulation sessions in text files with the date label. The user will be prompted to choose which variable (among the global ones of the simulation) to save and then indicate the name of the file (Figure 74): the *filename* can be inserted in the GUI window by the user, and for each file the user can specify which variable to save. The data visualization setup allows choosing what to display during runtime (but no interactions and zoom are available) and what to display at the end of the simulation (and these graphs will be fully interactive: the user will be able to zoom, select portion of the graphs, show values, and so on).

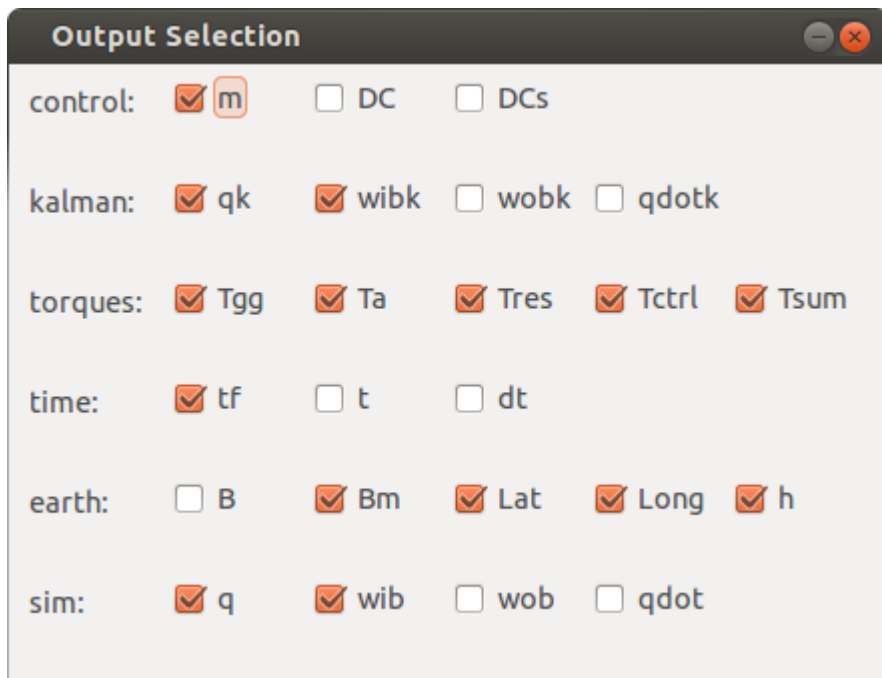


Figure 74: Outputs files definition window, sensible variables selection

3.4.6 Simulation execution

From the main window it is possible to start and stop the simulation through the button *RUN*. Other user interactions are not usually required during the runtime of the simulation but some small operation can be done. The pre-selected sensible data are plotted or tabled.

3.4.7 Results evaluation

At the end of the session, the user can load all the results of the last and/or previous simulation session through the button *ACCESS DATA* from the main window. All the files containing the simulation data are shown and the user can interact to produce graphs or tables and evaluate performance. The graphs are made by a free software (called *GNUPLOT* [18]) provided in any LINUX/UNIX release.

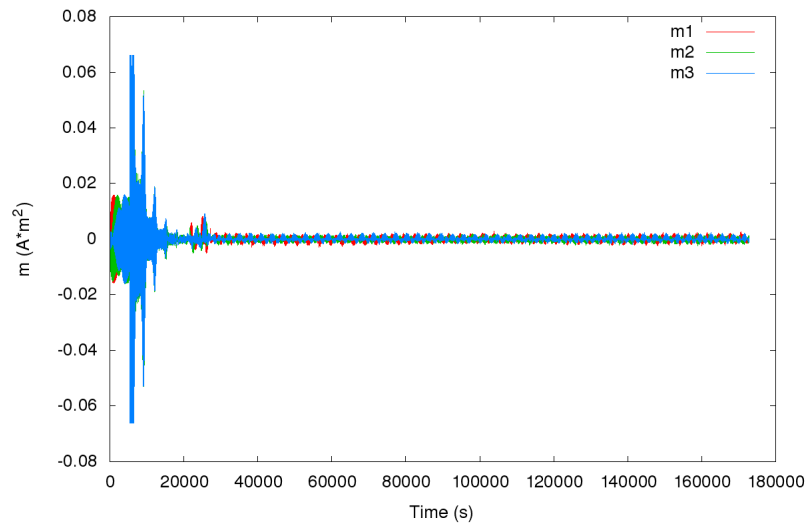


Figure 75: Example of a plot made after a simulation session with GNUPLOT

Chapter 3 reference

1. L.B.Rainey, *Space Modeling and Simulation.Roles and Applications Throughout the System Life Cycle*, The Aerospace Press El Segundo, California, 2004, American Institute of Aeronautics and Astronautics, Inc. Reston, Virginia
2. J. Eickhoff, *Simulating Spacecraft Systems*, New York: Springer, 2009. e-ISBN 978-3-642-01276-1
3. ECSS-E-TM-10-21A- *Space Engineering: System modelling and simulation* – (16/04/2010)
4. J.Connell and L.Shafer, *Structured Rapid Prototyping*, Yourdon Press, 1989.
5. L. Feruglio *Design of a Hardware in the Loop Simulator for Space Systems*, , MSc thesis, Politecnico di Torino, July2012
6. UNIX manual, <http://unixhelp.ed.ac.uk/CGI/man-cgi>
7. ECSS-E-ST-40C- *Space Engineering: Software*, 06/03/2009
8. ECSS-E-ST- 70- 31C *Space Engineering: Ground systems and operations - Monitoring and control data definition* – 31/07/2008
9. W.Fhese, *Automated Rendezvous and Docking of Spacecraft*, Cambridge University Press, 2003, ISBN 0 521 82492 3
10. Computer Science Corporation (Members of the Technical Staff Attitude Systems Operation), *Spacecraft Attitude Determination and Control*, J. R. Wertz, Ed. Dordrecht, The Netherlands: D. Reidel Publishing Company, 1978,
11. P.H.Zipfel, *Modeling and Simulation of Aerospace Vehicle Dynamics*, American Institute of Aeronautics and Astronautics, 2000, 1-56347-456-5
12. M.J.Sidi, *Spacecraft Dynamics and Control (a practical engineering approach)*, edited by M.Rycroft & R.Stengel, 2006, ISBN 0-512-55072-7
13. J. Wertz, D.Everett, J.Pushell,*Space Mission Analysis and Design*,4th Edition, Space Technology Library, ISBN 1-881883-10-8 , 2013
14. V. Chobotov, *Orbital Mechanics (3rd edition)*, AIAA Education Series,2002, ISBN: 1-56347-537-5
15. Bong Wie, *Space Vehicle Dynamics and Control*, Published by American Institute of Aeronautics and Astronautics, Inc., 2008, ISBN 978-1-56347-953-3
16. W.E.Hammond *Space transportation: a systems approach to analysis and design*, vol. 1, , AIAA Education series, 1999 ISBN 1—56347-472-7
17. J. Wertz, *Mission Geometry; Orbit and Constellation Design and Management*, Space Technology, Library, 2001. ISBN: 1-881883-07-8
18. <http://www.gnuplot.info/>

Chapter 4. The test case: e-st@r CubeSat

The AeroSpace System Engineering Team (ASSET) has been working on small satellite missions of Politecnico di Torino since 2003 [1]. One of the main research fields deals with defining and implementing effective and efficient methodologies to support the satellite development during all phases of its life cycle. To this purpose, we decided to develop an M&S-based tool to support and improve the AIV activities within a CubeSat project. In this chapter the description of the V&V process based on HIL architecture is provided. The HIL simulator developed within this PhD research has been used for a case of interest to demonstrate the feasibility of the approach. Moreover, main opportunities are highlighted together with challenges peculiar to CubeSat applications.

4.1 CubeSats

CubeSats are small satellites which share a common interface and equipment standardization [2]. The basic CubeSat is a 10cm-side cube-shaped platform whose mass is less than 1.33 kilograms [3]. By its nature, a CubeSat is a cheap spacecraft which can be produced in only a few months. Current CubeSat and small satellite missions are mostly developed for LEO (Low Earth Orbit) application, and the number of scientific goals/tasks that they can perform is still limited. CubeSats are nowadays a mature technology to perform Earth observations, and they are a valid educational tool to train young engineers and students in the process of conceiving, implementing and operating a space mission. CubeSats are also frequently developed in academia and by space agencies as way to build spacecraft in a quick and affordable way [4] [5]. For example, NASA Jet Propulsion Laboratory has recently developed M-Cubed/COVE [6] in cooperation with University of Michigan. COVE will validate an image processing algorithm designed to survey the impacts of aerosols and clouds on global climate change. NASA JPL is also designing LMRSat [7] (Low Mass Radio Science Transponder Satellite, 2U). NASA Goddard is currently developing HeDi (Helium Doppler Imager, 3U CubeSat) and TechCube 1 (technological demonstrator, 3U) [8]. In academia, MIT and Draper Lab are building ExoplanetSat [6], a 3U CubeSat that aims to detect SuperEarth exoplanets by the transit detection method. Another CubeSat being designed and tested at MIT is MicroMAS [10], which aims to use a spinning payload to study Earth's atmosphere. Other CubeSat missions for LEO have been recently developed at University of Michigan [11] (Radio Aurora Explorer, RAX), University of Colorado [12] (Colorado Student Space Weather Experiment, CSSWE), University of Hawaii [13] (UNP 6, Radar Calibration CubeSat), and others.

In Europe many universities have developed and are currently running CubeSat projects. In the last few years, the European Space Agency has carried out an important initiative that ended with the launch of 7 CubeSats from selected universities of the member States in February 2012 thanks to the new VEGA European Launch Vehicle [14]. Other universities in Europe have already launched their own satellites on commercial launchers, while others are ready to launch in the next few months. Most of the CubeSat today in orbit have primarily educational objectives and secondarily technological demonstration and/or scientific purposes [15]. After the successful first initiative, ESA is now promoting new CubeSat projects both in the education and in the scientific mission's areas. Two other programs currently under development are worth mentioning, both aimed at setting a constellation of CubeSats in LEO. QB50 is a FP7 international project coordinated by Von Karman Institute for Fluid Dynamics. Aim of the scientific mission is to study temporal and spatial variations of a number of key parameters in lower thermosphere (90-320 km) with a network of about 40 double CubeSats [16]. HumSat is an international project initiated by the University of Vigo, under the patronage of ESA and UNOOSA, with the objectives of monitoring climate changes and supporting humanitarian initiatives. The aim of the mission is to launch a CubeSat constellation for supporting a general-purpose

communication space-based service, above which the different users will be expected to build and use their own application [17] [18] [19].

CubeSats have been mainly developed for educational purposes, but they are increasingly being used for real science and service missions [20]. We are convinced that CubeSats can be useful for the attainment of a broad set of mission goals, including science, technology demonstration, communications, and Earth observation [21] [22]. In order to understand and eventually reach the full potential for CubeSats science missions, we must advance technology for almost all space satellite subsystems. Specifically, attitude control and navigation systems need to be improved, especially as the requirements for precise pointing and control for CubeSat become more stringent for communications at a distance and payload performance. Additionally, more advanced communication systems able to handle higher quantity of data and more distant transmissions need to be developed [23]. Propulsion is also an issue for advanced CubeSat and deserves particular efforts.

Finally, reliability of CubeSats is another area that demands improvements, because educationally-driven missions have often failed. The failure of CubeSats is dominated by infant mortality, which can be traced back to design weakness and/or ineffective Assembly-Integration-Verification (AIV) planning and execution. We believe that mission success is highly influenced by a robust design on the one hand, and by an effective verification and validation campaign on the other hand.

4.1.1 CubeSat verification process and HIL simulation

We think that an effective and exhaustive Verification and Validation (V&V) process may help to increase the reliability of small-scale satellites, as claimed also by Dubos et al. [24]. Talking about the V&V activity, it is worth mentioning that a major difference between CubeSats and traditional spacecraft exists. To date, the main objective of CubeSat V&V campaigns has been the demonstration of the satellite's capability to survive the launch phase without jeopardizing the launch vehicle and other spacecraft in the payload bay. This implies that CubeSats have been extensively tested against launch environment requirements upon request of launch authorities. Unfortunately, minor effort was committed to the verification of functional and operational requirements. These verifications are fully demanded to CubeSats developers, and no guidelines exist.

The International Organization for Standardization (ISO) is working on the definition of a new standard devoted to the verification of small-scale satellites. The standard is meant to improve the reliability of this class of satellites, while keeping its nature of low-cost and fast-delivery. In fact, some tests are necessary to improve the reliability of small-scale satellites to a level acceptable as a subject of commercial investment. At the same time, applying the same test requirements and methods as the ones for traditional large/medium satellites would kill the advantage of low-cost and fast-delivery. The new ISO standard will reflect the work done within the Nano-satellite Environmental Tests Standardization project described in [25]. The European Space Agency (ESA) has recently launched the *Fly Your Satellite!* initiative: selected university teams from EU and Canada are supported by ESA's experts for the development of their CubeSats. The project aims at increasing CubeSat mission reliability through several actions: to improve design implementation, to define best practice for conducting the verification process, and to make the CubeSat community aware of the importance of verification.

The activity discussed in this chapter is located within this framework. In particular, we focused our attention on improving the effectiveness of the functional verification of CubeSat. It is worth mentioning that for CubeSat programs the functional verification is usually carried out via simulation with no hardware in the loop. In most cases, CubeSats are tested against functional requirements in ambient condition, and then they undergo the environmental tests, followed each by a reduced functional test. Rarely CubeSat's functionalities are tested in orbit-like conditions mainly because of two reasons. First, this test would require resources, facilities and costs usually unsustainable for a

student initiative. The facilities devoted to these purposes are few and not appropriate for the test of so-small objects; they are expensive to hire, and inaccessible for unqualified users, i.e. the students. Second, it often occurs that the operational orbit is still unknown when the CubeSat is designed, integrated and tested. To reproduce physically the whole set of orbit conditions would result in a tremendous effort, in terms of both budget and testing time. The HIL approach represents the right balance between performance and cost of the testing activity.

From the available literature and the candidate's experience, HIL simulation is a methodology which has not been largely used for small satellites and in particular for CubeSats. HIL applications for small space platforms are quite recent and they are mainly oriented to the verification of Attitude Determination and Control System (ADCS) performances [26] or, more in general, the dynamic behavior of satellites [27]. CubeSat projects may benefit to a great extent from the application of HIL technology in V&V activity for the following reasons:

1. HIL simulation guarantees that we get the actual response from the real hardware included in the loop. HIL testing results much more authentic if compared with other simulation techniques. The presence of the hardware allows for example the accurate measurement of lag, signal noise, signal dropout, actuator saturation, and other parameters
2. Including the real hardware in the simulation loop simplifies the modeling activity. It may happen that accurate mathematical models of some devices are too complex to build or they are not readily available. This is particularly true in case of a CubeSat project, which makes extensive use of Commercial Off The Shelf components
3. Unpredictable interactions among pieces of hardware can be detected and eventually straightened out. A better understanding of the integrated system's behavior can be achieved early in the development process
4. The hardware can be easily tested in several orbit conditions, as these are virtually reproduced. This is particularly useful for CubeSat platforms which shall be flexible enough to adapt to different mission conditions.

Notwithstanding these opportunities, using HIL simulation for CubeSat projects entails a couple of concerns. First, the HIL approach necessitates test facilities equipped with flight hardware and dedicated support equipment. On the one hand, this means additional cost and longer test preparation time if compared with pure, AIL, or SIL simulations. On the other hand, the increased cost and time can be justified taking into account that HIL simulation yields greater insight into system's performance than other simulations. Second, traditional space systems have been tested via the HIL technique for decades in the industry. CubeSats are usually developed by universities, research centers, and small medium enterprises, which are not generally provided with certified simulators. Brand-new simulators require to be validated and the process may represent an issue for CubeSat programs, for example in terms of budget constraints and return of the investment. If the simulator's architecture is flexible enough, it can be used for several projects and not only for a specific mission, thus justifying the investment.

The abovementioned challenges could be overcome by the standardization of ground support equipment and facilities devoted to HIL testing of CubeSats. The cooperation within the CubeSat Community plays a major role in this scenario.

4.2 E-st@r Program overview

The e-st@r (Educational SaTellite @ politecnico di toRino) project is a structured hands-on education and research program based on CubeSats development. The main objective is to prepare future generation of space professionals. University programs shall take up the technological challenges issued by the scientific community and the industry, and they must help to improving knowledge necessary to build future space missions. The big challenge is doing space missions at low cost while

keeping reliability as high as possible. For these reasons, one of the main tasks of our projects is the development of methodologies to support both effectively and efficiently the entire satellite lifecycle. As Figure 76 illustrates, a space mission carried out by a University is driven first of all by the relevance that the mission has both for the research and the education purposes, being at the same time constrained by limited budget and resources.

The mission statement defines the high level objectives for a mission, i.e. why the mission is required and for which purposes the system that shall perform the mission exists. The mission statement for the e-st@r program sounds as follows: “*To educate aerospace-engineering students on systems development, management, and team work. To achieve insight in the development of scenarios and enabling technologies for future space missions.*”

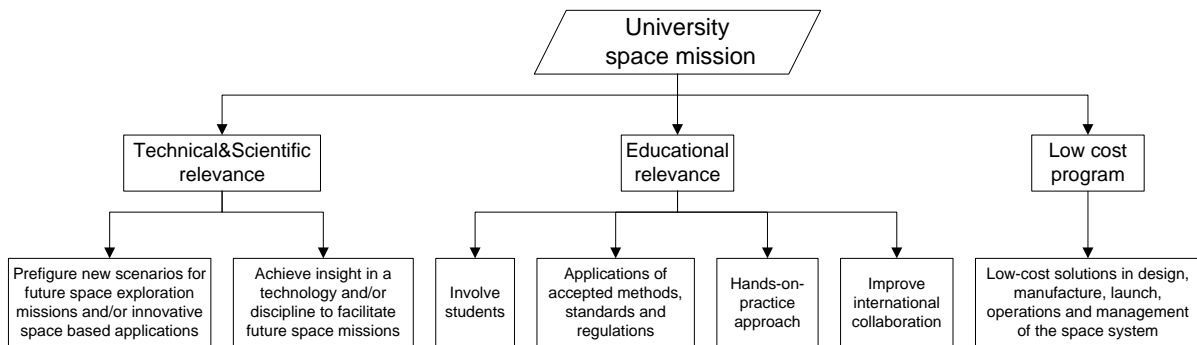


Figure 76: e-st@r program guidelines

The main program guidelines have been assumed as high level objectives and constraints for the e-st@r program. They can be listed as follows:

- **WHAT/1:** To inspire and prepare future space-professionals: students are the end users of the mission
- **WHAT/2:** To improve knowledge in space science and engineering: real world shall take advantages of our missions
- **WHY:** To meet stakeholders’ needs. Stakeholders are: students & civil society, scientific community, industry
- **HOW:** To carry out a space program from the design to in-orbit operations, completely managed by students

Taking into account these assumptions, the mission objectives and system requirements have been established, from which the technical specification derived, for both the space and the ground segments.

4.2.1 Mission objectives

The Mission Objectives (MO) represent the broad goals which the system must achieve to be *effective, productive, efficient* and *useful*. In our case the system is a CubeSat-class satellite, developed by university students. The motivations that led us to the definition of the mission objectives are therefore to be found within the university setting, and include "needs" of both scientific-technological importance as well as educational significance.

At the present moment, we are experiencing a “big” revolution in CubeSat Community because a great interest in this kind of space platforms is growing also among actors other than universities. For few years after their invention, CubeSats have been developed exclusively within the academia for higher education purposes with possible technological and scientific secondary objectives. We can now say that nowadays the interest in CubeSat missions for scientific or commercial services is growing

steadily and CubeSats are thus evolving from pure university education tools to spacecraft buses for scientific and commercial payloads.

Notwithstanding the necessity of keeping cost down and taking into account the educational purpose of the CubeSat program, our CubeSat missions also have scientific and/or technological objectives, which reflect real interests of the scientific and industrial communities.

One of the most significant challenges is how to accomplish science goals while facing severe limitations on mass, volume and power. On the other hand, one of the benefits of CubeSats is the relatively low cost from standardized components and piggy-back launch opportunities. To accomplish more ambitious scientific goals, a certain set of technological challenges need to be addressed. We strongly believe that CubeSats can contribute to broad science goals, if supported by the appropriate set of technologies.

The capability of autonomous attitude determination and control is one of the enabling technologies for future CubeSat missions, specifically where requirements in terms of stabilization and pointing accuracy are critical to the effectiveness of experiments, payload operations, communications, and in turn to the mission success. More than 100 CubeSats have been launched since year 2003, about 45% of them hosting an Active Attitude Determination and Control System (A-ADCS) on-board⁴. The remaining part includes either satellites with no actuation or with passive control systems, the latter often providing only few pointing options, weak accuracy and very limited attitude maneuverability.

The will to keep on focusing on attitude control systems and the purpose to enhance A-ADCS capabilities for CubeSats have been well supported by the attention and the curiosity of the students in our university. Moreover, the scientific community and the industry have proved to be interested in this topic in several occasions. For these reasons the primary scientific objective of e-st@r missions is:

- to demonstrate the capability of autonomous determination, control and maneuver, through the development and test in orbit of an A-ADCS entirely designed and manufactured by students.

We established the *effectiveness* of the system in terms of stabilization, pointing, slew maneuvers and response-time requirements to be verified in order to achieve the mission success.

Besides, we identified the way for the system to be *productive*, in terms of quantity and quality of data that shall be collected and analyzed, in order to demonstrate the effective capability previously defined. In this regard, the possibility to rely on the largest community of CubeSat and telecommunications enthusiasts, as the radio-amateurs community is, plays a key role in support of this demonstration.

The secondary objective concerns the possibility of:

- testing in orbit COTS technology and self-made hardware.

It describes both the *usefulness* and *efficiency* attributes of the mission. The former relies on the educational aspect, as well as on the possibility given to the team to enhance its knowledge and experience. Under a different perspective, the mission can be seen as a test-bed for components, as well as an opportunity to perform reliability studies.

The latter, *efficiency*, refers to the capability to develop a complex system with few resources. It is mainly related to the university low budget constraints, but it turns into an opportunity for bearing in mind cost reduction and simplicity at any level of the engineering process.

The motivations, needs and constraints that led to the definition of the mission objectives are summarized hereafter and shown schematically in Figure 77:

- Technology advances: CubeSats' science missions need stabilization, pointing accuracy and maneuver capability not achievable by passively stabilized satellites

⁴ Data based on public information available and updated to May 2013.

- Team skills and interests: 1) students are usually skilled in simulations and enjoy doing tricky calculations, 2) attitude dynamics is one of the most appreciated subjects in aerospace students' curricula at our university
- Program budget: the cost of the payload shall be limited. Simple and reliable technologies shall be included in the project, and standard materials shall be used. COTS components shall be preferred for cost containment and procurement time
- Methodologies and tools: the development of the system shall be based on simulations and analyses (mathematical models, CAD models, etc.: many resources at university) rather than on production and testing (lack of facilities at university)
- Orbit and launch vehicle: orbit parameters and launcher data are not yet defined
- Limited mass, size and power: 1U CubeSat platforms have strong limitations in mass, size and power availability
- Development time: launch date is uncertain so CubeSat developers shall be ready to deliver the CubeSat upon request within a limited period of time. The original program schedule depicts a short duration program and limits the development, test and verification time to a few months.

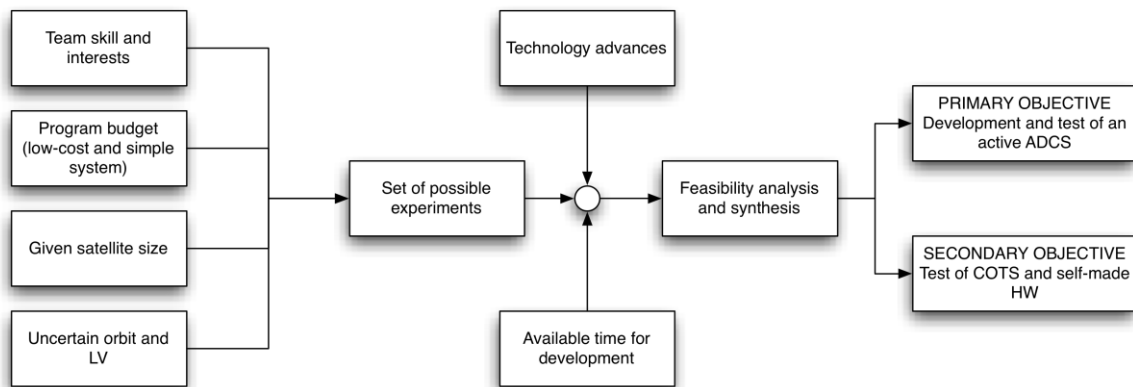


Figure 77: e-st@r mission objectives

This chapter describes the design, development and verification of the CubeSat A-ADCS using the methodology and the StarSim simulator in any phase of the e-st@r CubeSats life cycle.

E-st@r-I is the first element of the CubeSats family we are working on. It is a 1U satellite developed by students with the aim of testing in space new technologies and design solutions. The CubeSat was chosen by ESA within the “Educational Payload on the Vega Maiden Flight” initiative. The launch took place in 2012, February 13th, from Europe’s spaceport in French Guyana. The second 1U CubeSat (e-st@r-II) is under test at the Systems and Technologies for Aerospace Research Laboratory (STAR Lab) of the Department of Mechanical and Aerospace Engineering. E-st@r-II is in the ESA’s *Fly Your Satellite!* program that would lead up to a Launch opportunity in the near future.

4.2.2 Mission architecture

Basic CubeSat mission usually consists in a space platform orbiting the Earth at low altitude and one or more ground stations gathering/collecting data and sending commands from/to the spacecraft.

One peculiar characteristic of most CubeSat missions is that spacecraft are launched piggyback as secondary payload, and the orbit is imposed by the primary payload and the launch vehicle. Furthermore, there are cases, such as FYS initiative, in which the orbit is unknown at the beginning of the project (when the mission is designed) because the launch opportunity is still to be found.

Another factor to be considered when designing a CubeSat mission is that the project is handled by students. In particular, mission operations cannot rely on a 24/7 dedicated team, so a high grade of autonomy is desired.

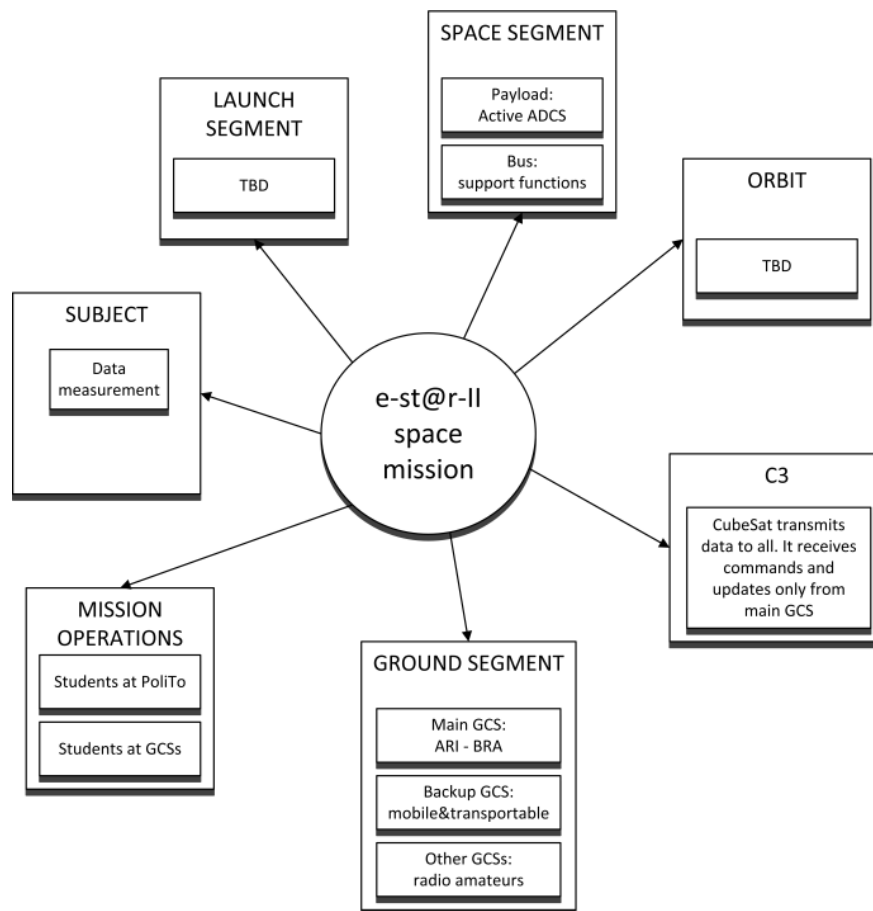


Figure 78: Mission architecture (e-st@r-II mission)

Elements of the e-st@r mission architecture are illustrated in Figure 78:

- Space segment:** 1U CubeSat, payload and bus. The platform includes the Electrical Power Subsystem (EPS) devoted to provide, store, control and distribute the electrical power on-board, the Communication Subsystem (COMSYS) that provides the interface between the space and ground segments, and the On-Board Computer (OBC) which handles and executes commands, manages and stores data and performs autonomous on-board operations. The structure/mechanical subsystem (S&M) is devoted to carry the loads induced by the launch vehicle, to support and protect all other spacecraft subsystems. Passive thermal control has been designed for the CubeSat. The payload of e-st@r CubeSats is an active attitude determination and control system based on inertial and magnetic measurements, with magnetic actuation. The system shall provide the desired (nadir) antenna pointing and/or proper reorientation maneuvers when required with commands from GCS.
- Ground segment:** The Ground Segment of the e-st@r project consists of two ground stations: the main Ground Control Station (GCS) is the ARI – section of Bra (Ham Radio Club – IQ1RY) and the second is the Mobile Ground Control Station (MGCS) located at Politecnico di Torino which is transportable and may be transferred everywhere. ARI-Bra station (Figure 79) is an existing radio amateur station that supplies all the elements needed to communicate with e-st@r satellites: it is able to send commands to the satellite and to receive the telemetry

packets. The CubeSat Team tracked and communicated with e-st@r-I CubeSat from this station. Other stations around the world (radio amateur network) may receive the CubeSat signal, but cannot command it.

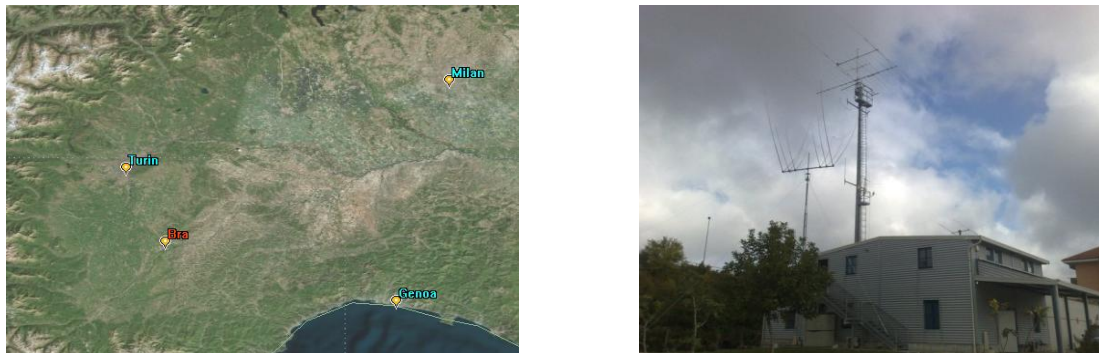


Figure 79: Main GCS location (left), building and antennas (right)

- **Launch segment:** piggy back launch
- **Subject:** data measurement. The thing that is sensed by the payload is essentially the LEO environment, in terms of magnetic field. Data from magnetometer are used for the attitude determination task and successively exploited for the control task. Moreover, the satellite shall provide a large quantity of data about his orientation and angular velocities. This data will be subject of analysis on ground to verify the determination and control algorithms. Other telemetry data (i.e. temperatures, battery state of charge, on board computer status, communication system status) will be collected and analyzed on ground to verify the proper functioning of COTS components, self-made hardware and software.
- **Orbit:** direct injection into LEO
- **Operations:** students at main and backup GCSs. Data processing at STARLab for deeper investigation, or in case of emergency. Radio-amateur network and CubeSat Community will be involved in the data collection and will support the mission operations.
- **Communications:** Telemetry data are sent by the CubeSat to the main GCS and to other stations all over the world. Data are coded in a defined protocol and transmitted to ground. The uplink commands are received on board and relayed to other subsystem (mainly to OBC) according to a defined protocol. Note that the two mentioned protocols are not necessary the same: in fact, in order to reach the largest quantity of ground stations and gather most data as possible, the downlink protocol is public. Information about frequency, antenna, modulation, protocol, destination and source, call sign, etc. will be published on the project's website and available for the radio-ham community. The uplink protocol instead, should contain safeguards against intentional or unintentional signal corrupting the command link, or unauthorized commands from being transmitted and accepted by the satellite: for this reason commands links is encrypted with secure code and format will not be published.

4.2.3 Mission phases

E-st@r-I and e-st@r-II missions are very similar, as well as the two CubeSat are. Some minor differences exist, that will be highlighted when needed. As an example, the mission profile of the e-st@r-II CubeSat is depicted in Table 32, which includes only orbit lifetime. The detailed description of each phase (including ground operations) and related operative modes are described below.

Phase/Event	Duration	Remarks
Launch	T0	T0 is the actual lift-off time of the LV
CubeSat release and activation	T1	T1 is given by the burnout time necessary to reach the orbit plus the time needed to be ejected from the deployer. The CubeSat is instantaneously activated by the DS and enters the first mission phase in orbit.
CubeSat appendages deployment	T2	T2 is given by the time delay imposed by CDS (30 min + margin) plus the time needed to deploy the antenna (approximately 1 minute).
Commissioning	T3	T3 is the time needed to prepare and check out the CubeSat for nominal operations. T3 ranges from a minimum of 10 minutes to several days, depending on commissioning activity result.
Begin of mission	T4	T4 is the time at which the nominal mission begins officially.
End of mission	T5	T5 = 1 (TBC)-to-12 (TBC) months after T4. Extended duration shall be considered.
Disposal of CubeSat	T6	T6 is the time needed for the orbit to decay after the mission has been declared finished (after which the CubeSat will burn in the upper layer of Earth's atmosphere).

Table 32: Mission profile (ref. e-st@r-II mission)

4.2.3.1 *Launch and Early Orbit Operations - LEOP*

LEOP refer to launch, CubeSat release and activation and CubeSat appendages deployment. During the launch phase (i.e. from lift-off to the deployment in orbit), the CubeSat shall be switched off as specified in the CDS (TBC). The launch phase is characterized by relevant vibro-acoustic and thermal environments. The CubeSat is designed to survive the launch phase. A proper “dormant” operative mode has been implemented for this phase.

When the orbit is reached, the CubeSat is released and its orbit life starts. The CubeSat is immediately activated after deployment because the DS closes the relevant electrical circuit, but it remains in a stand-by state for 30 minutes at minimum (TBC). During this period of time, no RF emissions are allowed (TBC). The injection into orbit may add some rotational velocity to the CubeSat.

The deployment of the antenna is commanded by an automatic procedure executed by the OBC. After the deployment of the antenna occurs and has been checked out by the OBC, the launch and early operations continue with the automatic activation of the COMSYS to set up initial parameters and start the downlink communication.

4.2.3.2 *Commissioning*

When the antenna is finally deployed and COMSYS is active, the early mission operations begin with a phase called commissioning. During this phase, which may be actually quite short (in the order of days), the CubeSat is operated from ground to check its health. During commissioning, the operations team analyses the telemetry received from orbit and decides how to operate the CubeSat depending upon the results of the analysis. The activation of the CubeSat to reach full capability is done on an incremental-based approach. If the phase successfully closes out, the payload can be activated and the nominal mission begins.

4.2.3.3 *Nominal mission operations*

Once the commissioning phase is successfully completed, the CubeSat enters its nominal mission in a basic mode of operation, consisting of downlink and uplink communications. During this phase, the CubeSat is ready to operate its payload, the A-ADCS.

The CubeSat can receive commands from the main GCS to change its operative mode, or simply for updating parameters. Among the commands, some are devoted to payload control. The A-ADCS can operate in three modes:

- *Mode 0 - standby*: the microcontroller (ARM9) is the only active component;
- *Mode 1 - determination*: IMU and magnetometer are active in order to evaluate the angular velocities and attitude;
- *Mode 2 - control*: the controller commands the MTs to de-tumble/stabilize the CubeSat.

When the A-ADCS is activated in Mode 1 or Mode 2, the CubeSat enters its Full Mission mode. The A-ADCS can either de-tumble the CubeSat or stabilize it around a desired attitude, depending on the initial rotation velocities of the satellite.

From simulations results, it is known that the de-tumbling phase may last from 90 to 100 minutes (about one orbit), depending upon initial angular velocity, environmental parameters variation, as well as on accuracy of the dynamics and control models. The de-tumbling phase ends when the angular velocities are less than a threshold defined in the system specification. At the end of this phase the CubeSat is quite stable. The CubeSat can be maintained in the desired attitude or a slew maneuver can be commanded to change its attitude. These results are described later in this chapter into the detail.

4.2.3.4 *End-of-Life*

Once the CubeSat has accomplished its mission, approximately after 1(TBC) to 12(TBC) months from launch, it will remain in orbit until the natural decay occurs. From early estimation, it results that it should take about 12 years for the orbit to decay.

The propagation has been performed with a reference circular LEO orbit, 600 km altitude, 97.8 degrees of inclination. Results may vary depending on the orbital parameters and on the atmosphere model used. Detailed results can be obtained only upon the definition of the final orbit.

During the disposal phase, the CubeSat communications may be shut down upon request of FCC. In this case the CubeSat will be commanded from ground to switch to a degraded mode of operation.

4.2.4 **Modes of operation**

In order to accomplish its mission, the CubeSat shall be able to operate in different modes depending upon the mission phase and operational needs. Taking into account all the applicable requirements and the mission phases, it is possible to derive the necessary operative modes. In particular, the potential failures and malfunctions have been considered. Moreover, transitions between modes of operation have been implemented, and they can be either automatic or commanded from ground.

The operative modes which have been designed and implemented are described Dormant Mode

From CDS requirements, the necessity to deactivate the CubeSat emerges. As a consequence, a corresponding operative mode has been designed and it is named “Dormant Mode”. When in dormant mode, the CubeSat is completely turned off, no subsystem is active.

One option in order to meet CDS requirements is to launch with discharged battery/ies. The second possibility is to deactivate the satellite by means of one or more devices. Option 1 poses a serious hazard to the mission completion. In fact, in case the battery charging operation is not executed when the satellite is in orbit, no power is available at all, not even for a short period of time. To be launched with charged batteries guarantees that the satellite can work at least for a limited period of time, given by the batteries state of charge at the moment of the launch. In this case (Option 2) a deactivation

system is needed to satisfy the applicable requirements. Applicable requirements from CDS dictate that (at least) one DS shall be included in the design. The switch can be used to maintain the CubeSat in the dormant state. Moreover, they imposes that a RBF pin is included in the design to cut all power to the satellite once it is inserted in the system, but this pin shall be removed before launching.

The functions carried out by the two switches are similar but not identical, and moreover, their use is completely different. In fact, the DS is included in the CubeSat in orbit configuration, while the RBF pin (as the name implies) is not. From the electrical point of view, their role is slightly but significantly different. They both cut the CubeSat power off by interrupting the proper on board circuits, but 1) the DS acts on the CubeSat power distribution bus, i.e. it prevents the electrical current to be distributed to the subsystems, while 2) the RBF disconnects the battery bus, isolating the power generation from the rest of the system.

On ground, it is possible to use both switches to turn off the satellite, but in orbit only the DS can be employed to the same purpose. One switch (the DS) shall be sufficient to cut off the electrical power. When the DS and/or RBF are inserted, the system is not active because no power is supplied to the subsystems.

The e-st@r-II CubeSat will be launched with its batteries at the highest possible state of charge, and a deactivation system is included in the design (the same design was adapted for e-st@r-I). The system is composed by one electromechanical switch (the DS) able to cut the power off when it is inserted (when its metal label is pressed, then the electrical circuit is maintained open, and the distribution bus is disconnected from the system). The DS is inserted when the CubeSat is in the deployment system by means of a support which restrains the DS in the inserted position. When in the deployer on the LV, the DS alone shall guarantee the deactivation of the CubeSat.

When the CubeSat is on-ground, the RBF switch can be used to turn off the CubeSat. This is useful in particular during operations such as transportation of the CubeSat, assembly and integration in the deployment system, and for safe handling in general. The RBF pin shall be removed before launch, but also for maintenance on ground (e.g. for battery charging, or functional check out). It can be said that when the RBF pin is inserted in the satellite, the satellite itself is in a special “dormant mode”. It is desirable that the satellite is kept in this state during storage, in particular if a long period of storage is considered. It is worth mentioning that in this configuration the batteries are disconnected from the system, and so no discharge may occur, apart from the battery nominal self-discharge. With the RBF pin inserted in the CubeSat, the state of charge of the batteries can be maintained at the highest possible level.

The dormant mode shall be maintained in particular for the operations on ground (excluding test, maintenance and checkout operations) and during the launch.

Figure 81 shows the switch configuration for the EPS on board of e-st@r-II. “Pull pin” refers to the RBF switch, while “Separation Switch” refers to the DS

The figure reports also the associated mission phases. It shall be noticed that in case some failures occur, the satellite can operate in degraded modes.

Figure 80 shows the diagram of the mission phases and modes of operation of the satellite during each phase.

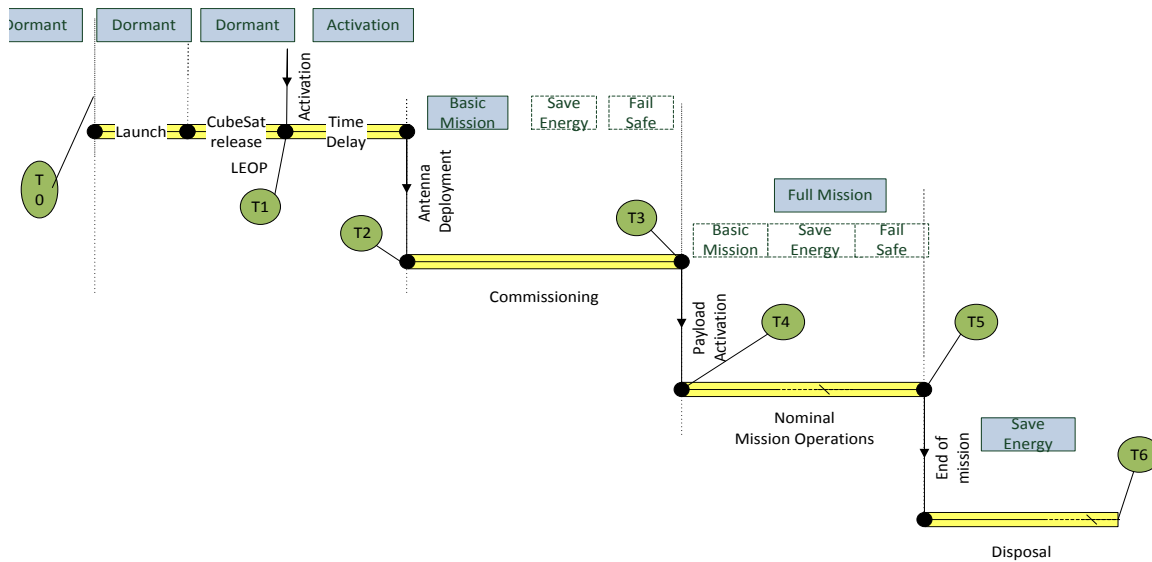


Figure 80: Mission phases and operative modes

4.2.4.1 Dormant Mode

From CDS requirements, the necessity to deactivate the CubeSat emerges. As a consequence, a corresponding operative mode has been designed and it is named “Dormant Mode”. When in dormant mode, the CubeSat is completely turned off, no subsystem is active.

One option in order to meet CDS requirements is to launch with discharged battery/ies. The second possibility is to deactivate the satellite by means of one or more devices. Option 1 poses a serious hazard to the mission completion. In fact, in case the battery charging operation is not executed when the satellite is in orbit, no power is available at all, not even for a short period of time. To be launched with charged batteries guarantees that the satellite can work at least for a limited period of time, given by the batteries state of charge at the moment of the launch. In this case (Option 2) a deactivation system is needed to satisfy the applicable requirements. Applicable requirements from CDS dictate that (at least) one DS shall be included in the design. The switch can be used to maintain the CubeSat in the dormant state. Moreover, they impose that a RBF pin is included in the design to cut all power to the satellite once it is inserted in the system, but this pin shall be removed before launching.

The functions carried out by the two switches are similar but not identical, and moreover, their use is completely different. In fact, the DS is included in the CubeSat in orbit configuration, while the RBF pin (as the name implies) is not. From the electrical point of view, their role is slightly but significantly different. They both cut the CubeSat power off by interrupting the proper on board circuits, but 1) the DS acts on the CubeSat power distribution bus, i.e. it prevents the electrical current to be distributed to the subsystems, while 2) the RBF disconnects the battery bus, isolating the power generation from the rest of the system.

On ground, it is possible to use both switches to turn off the satellite, but in orbit only the DS can be employed to the same purpose. One switch (the DS) shall be sufficient to cut off the electrical power. When the DS and/or RBF are inserted, the system is not active because no power is supplied to the subsystems.

The e-st@r-II CubeSat will be launched with its batteries at the highest possible state of charge, and a deactivation system is included in the design (the same design was adapted for e-st@r-I). The system is composed by one electromechanical switch (the DS) able to cut the power off when it is inserted (when its metal label is pressed, then the electrical circuit is maintained open, and the distribution bus is disconnected from the system). The DS is inserted when the CubeSat is in the deployment system

by means of a support which restrains the DS in the inserted position. When in the deployer on the LV, the DS alone shall guarantee the deactivation of the CubeSat.

When the CubeSat is on-ground, the RBF switch can be used to turn off the CubeSat. This is useful in particular during operations such as transportation of the CubeSat, assembly and integration in the deployment system, and for safe handling in general. The RBF pin shall be removed before launch, but also for maintenance on ground (e.g. for battery charging, or functional check out). It can be said that when the RBF pin is inserted in the satellite, the satellite itself is in a special “dormant mode”. It is desirable that the satellite is kept in this state during storage, in particular if a long period of storage is considered. It is worth mentioning that in this configuration the batteries are disconnected from the system, and so no discharge may occur, apart from the battery nominal self-discharge. With the RBF pin inserted in the CubeSat, the state of charge of the batteries can be maintained at the highest possible level.

The dormant mode shall be maintained in particular for the operations on ground (excluding test, maintenance and checkout operations) and during the launch.

Figure 81 shows the switch configuration for the EPS on board of e-st@r-II. “Pull pin” refers to the RBF switch, while “Separation Switch” refers to the DS

Mode	Mission phase	Description
Dormant	Launch and CubeSat release	The CubeSat is “dormant”, no RF emissions, no power consumption. All the subsystems are turned off
Activation	CubeSat appendage deployment	The CubeSat is activated by the DS. EPS is active. ADCS is in Mode 0: only the ARM9 microprocessor is active. The OBC starts booting and remain in a stand-by mode until all necessary checks are passed. Then the antenna is deployed. This mode is irreversible and cannot be repeated after the antenna deployment.
Basic Mission	Commissioning Nominal mission	The CubeSat sends telemetry packets to ground stations all over the world every 120 seconds. It may receive commands from main GCS and execute them.
Full Mission	Nominal mission	The CubeSat operates the payload, i.e. it actively controls its attitude when commanded to.
Fail Safe	Commissioning Nominal mission	This is an off-nominal operative mode, and it is used in case communication between OBC and COMSYS fails. In this case, COMSYS autonomously sends a Morse code (CW) every 5 minutes
Save energy	Commissioning Nominal mission End-of-life Upon request of FCC	This is an off-nominal operative mode, and it is used in case low power is detected on-board. The CubeSat only carries out vital function at minimum power consumption upon command from ground. Communications to Earth are limited to some extent and eventually they can be totally stopped. This mode also can be used in case a shutdown command is sent to the CubeSat upon request of FCC

Table 33: Operative modes

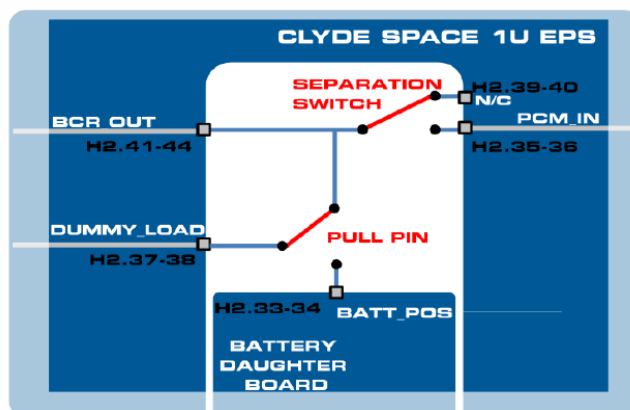


Figure 81: Switch configuration in the EPS board. Credit ClydeSpace Ltd.

4.2.4.2 Activation Mode

As soon as the CubeSat moves from its restrained position in the deployer system, the DS lets the power bus to be powered by the EPS (connection to PCM_IN, Figure 81). All the subsystems can be supplied with the electrical power they require. Automatically the OBC, and COMSYS are powered, the processor of ADCS is active, no sensors or actuators are active, but their activation is controlled by the OBC. In particular, it shall be avoided that the antenna deploys and communications start before all the conditions are verified.

It is clear that a software sequence shall be designed, in order to meet the requirements and to guarantee a safe CubeSat activation.

4.2.4.3 Nominal Mission Modes

Two nominal operative modes are defined for orbit operations. The following description refers to e-st@r-II. E-st@r-I had slightly different nominal operative modes wrt e-st@r-II.

- *Basic mission.* The CubeSat sends a telemetry string every two minutes (but the time interval can be changed from ground). The message encompasses data from which the health status of the satellite can be derived (temperatures, voltages, currents, subsystems status). The message can be received by all the radio-amateur stations on the assigned frequency. The protocol of the string is public. EPS, OBC and COMSYS work in their standard nominal modes.
- *Full mission.* The CubeSat is commanded to operate the payload. The A-ADCS can be operated in several ways to test its capabilities. First, the de-tumbling phase is tested, i.e. the ADCS is commanded to reduce the angular velocity of the CubeSat to a desired value. Then, the stabilization phase starts by changing the control strategy from coarse to fine pointing. Upon command, a slew maneuver can be requested. In this case, the desired final attitude is communicated to the CubeSat that shall compute autonomously the necessary maneuver to reach and maintain the objective attitude. The CubeSat sends a telemetry string every two minutes (but the time interval can be changed from ground). The message encompasses housekeeping telemetry (as in the basic mode) plus ADCS related data (angular accelerations and velocities, quaternion, magnetic field vector, magnetic torquers consumption). The message can be received by all the radio-amateur stations on the assigned frequency. The protocol of the string is public. The EPS, OBC, COMSYS and ADCS work in their standard nominal modes.

During both basic and full mission modes the CubeSat can receive commands from the main GCS (the ARI-BRA station), and from the backup station (the MGCS).

4.2.4.4 *Save Energy Mode*

During the orbit lifetime, some failures may occur to the CubeSat. In particular, a major failure is the loss of electrical power. This may happen for different reasons, some of which cannot be handled by the system. In case the power loss is due to a peak power consumption, a *save energy mode* has been implemented. When the battery bus voltage drops below a certain value, the satellite enters, upon command from the GCS, the save energy mode, in which it reduces (and eventually stops) communicating with the GCS and stops payload operations, until the bus voltage is restored to the desired voltage.

4.2.4.5 *Fail Safe Mode*

The *fail safe mode* has been thought and implemented to face the case in which the communication between OBC and COMSYS fails. This malfunction could be due to failed communication serial link between the two mentioned subsystems and this inhibits the telemetry downlink.

When this occurs, the COMSYS automatically switch to a Continuous Wave transmission (CW) consisting of a “estarc2” string translated in Morse-code. This transmission is totally independent from OBC functioning.

This mode of operations let the user on ground to know that a malfunction is localized on the OBC. Moreover, it allows the mission operators on ground to continuously track the satellite even if the telemetry is not available.

This mode of operation is activated only automatically when the communication link between the two mentioned subsystems is interrupted. The satellite can get out of this mode automatically if the OBC starts again the communication with the COMSYS, or if commanded from ground to do so. This mode does not affect the mode of operation of the other subsystems on board.

4.3 CubeSat design

The e-st@r CubeSats are characterized by the same system architecture: they are equipped with a set of subsystems for supporting the mission. The payload of the first two CubeSats is constituted by an Active Attitude Determination and Control System (A-ADCS). The attitude is determined by algorithms and Kalman filter in the A-ADCS software using data sensed by an Inertial Measurement Unit (IMU), and a 3-axes magnetometer. The actuation is guaranteed by three magnetic torquers (MT). The A-ADCS is controlled by a dedicated micro-controller (ARM9), on which the algorithms for determination and control run [28]. Details on this subsystem are given in next section.

The spacecraft bus is constituted by the Electrical Power System (EPS), the Communication System (COMSYS), the Onboard Computer (OBC), and the Structure and Mechanism System (S&M). The EPS collects and transforms solar radiation energy by means of five solar panels made of GaAs cells. Electrical power is stored in two battery packs, then regulated and distributed to other subsystems through two power buses at 3,3 Volt and 5 Volt. The COMSYS is a in-house design and manages downlink and uplink signals. It exchanges information with the OBC by a piece of equipment based on commercial components: a PIC16 modulates/demodulates and checks the signal, a radio transceiver amplifies the signal and a dipole antenna transmits/receives the signal to/from Earth. The OBC performs all onboard activities related to the command and data handling functions, the data storage in the SD memory card, and time synchronization. The structure is an aluminum alloy box designed to endure launch loads, and to host and protect the electronics from the harsh space environment. It also includes the deployment system of the antenna.

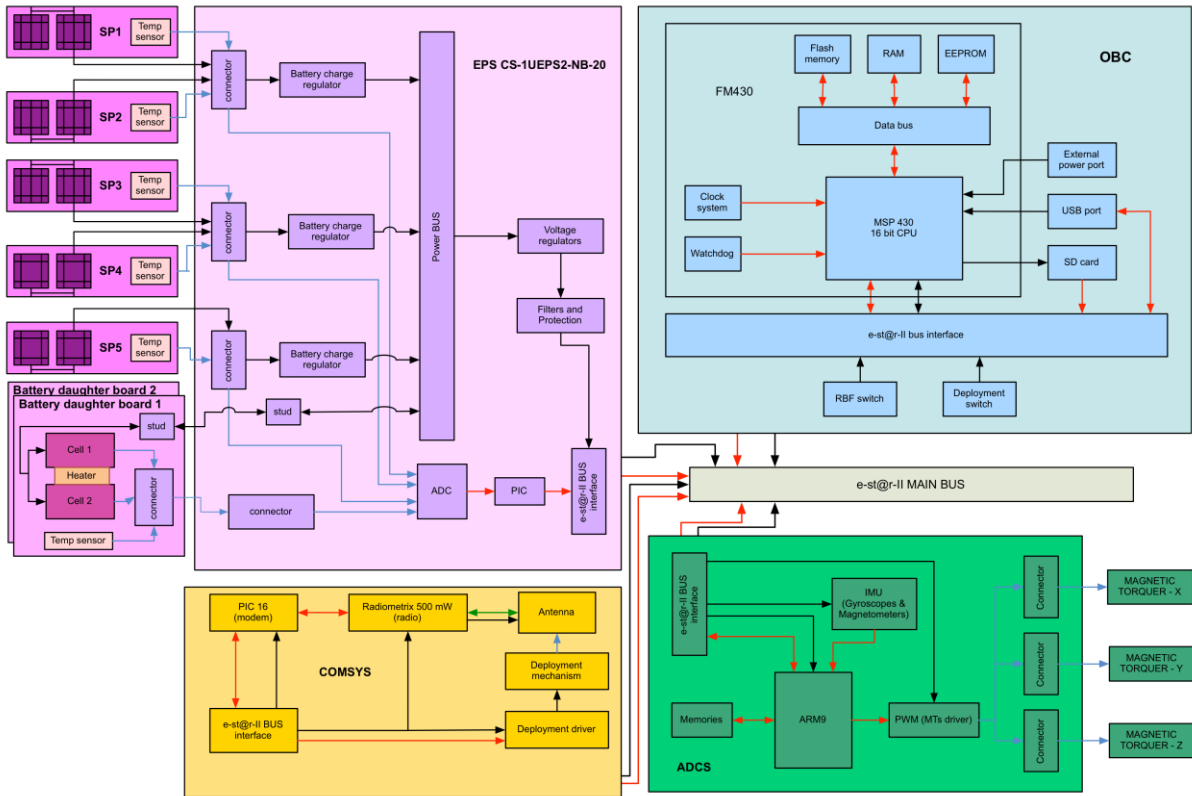


Figure 82: e-st@r blocks scheme

The system blocks scheme is shown in Figure 82. The main bus represents the common interface among subsystems, for both the data flow and the power line. A picture of the e-st@r-I flight unit is shown in Figure 83.

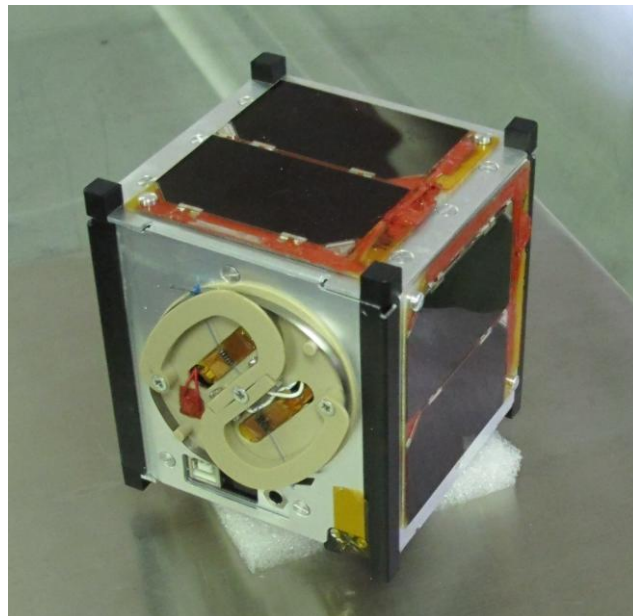


Figure 83: e-st@r-I flight unit

4.4 A-ADCS design and verification

The A-ADCS determines and controls the CubeSat’s angular orientation. The system must be able to ensure desired antenna pointing (to the nadir) and adequate reorientation when required, so an active control system is required because the pointing accuracy provided by passive systems could not be suitable for this mission.

4.4.1 Design of the A-ADCS of e-st@r CubeSats

The first step of the A-ADCS design is the analysis of needs and top level (mission) requirements, see paragraph 4.2. Reduced budget, reduced power, the small satellite dimensions and weight limit the design choice and impose to study low cost solutions without giving up performances and reliability.

Figure 84, Figure 85, Figure 86, Figure 87, and Figure 88 show the functional analysis, made through the functional tree approach, which allows defining the main subsystem functions:

- to determine the satellite angular data;
- to generate profiles for attitude
- to calculate the attitude maneuvers
- to control attitude (counteracting disturbance torques);
- to manage the operative modes;
- to manage own failures;
- to exchange data with other subsystems

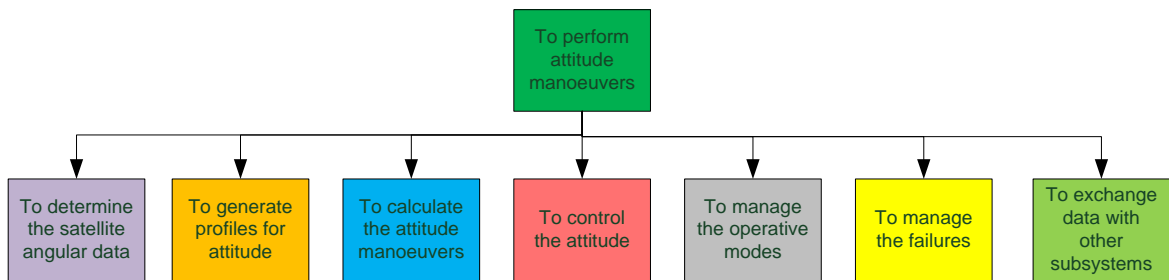


Figure 84: Functional tree (top level) of E-ST@R A-ADCS

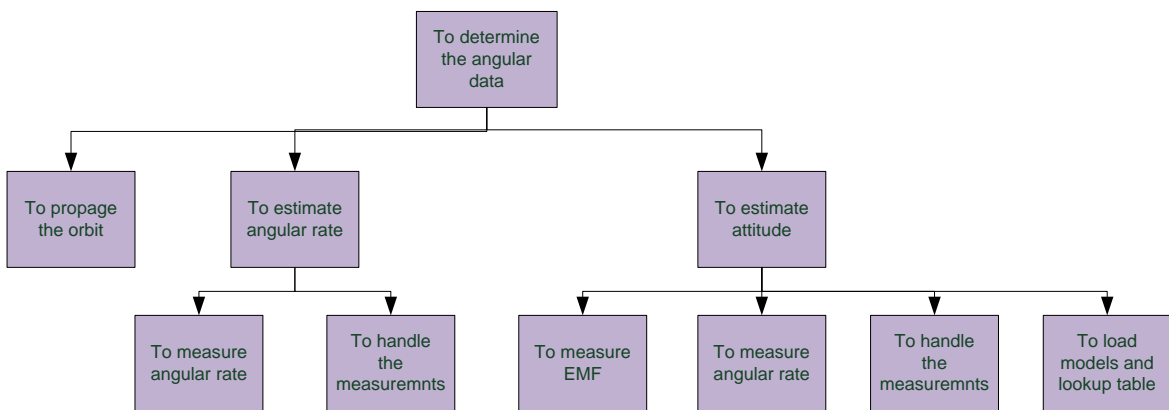


Figure 85: Functional tree (part 1) of E-ST@R A-ADCS

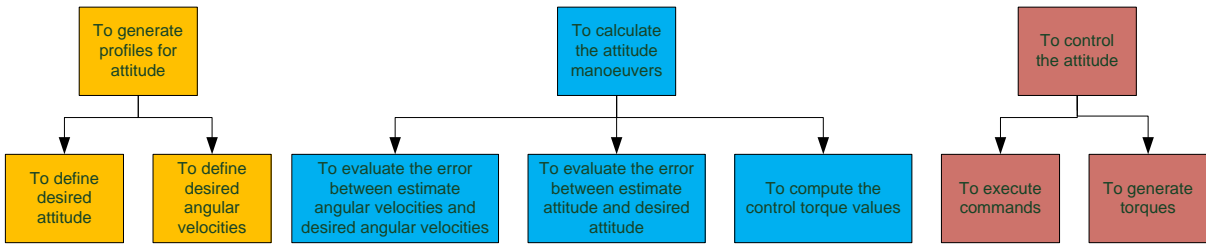


Figure 86: Functional tree (part 2) of E-ST@R A-ADCS

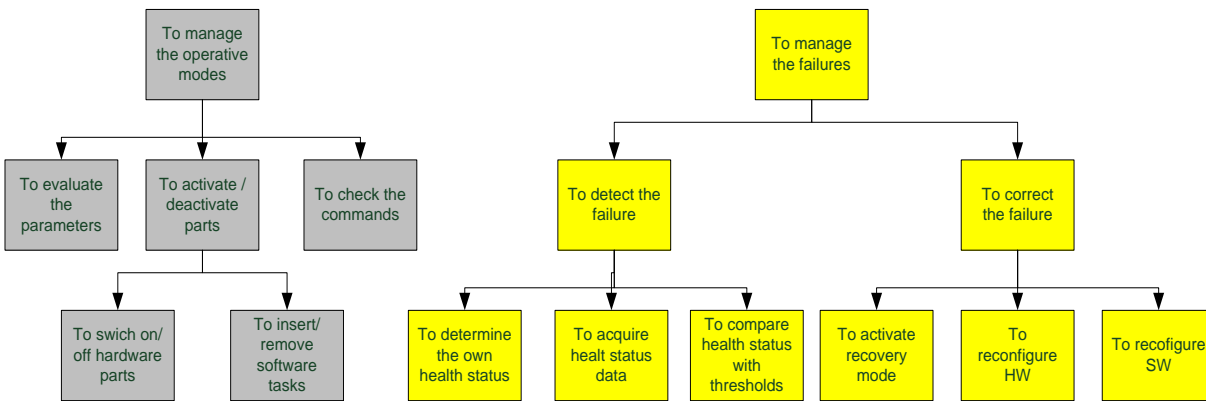


Figure 87: Functional tree (part 3) of E-ST@R A-ADCS

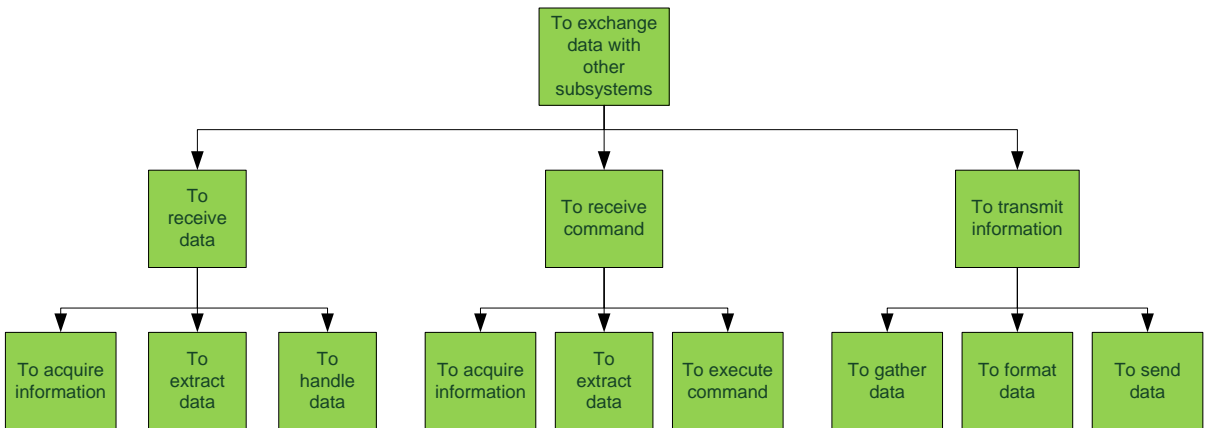


Figure 88: Functional tree (part 4) of E-ST@R A-ADCS

Figure 89 shows the interactions among the A-ADCS and other on board subsystems. EPS has to provide regulated voltages and currents according to the A-ADCS components. At the same time, the electrical loads shall remain within the power consumption imposed at system level for each subsystem. OBC communicates the new parameters and new commands to A-ADCS which shall provide the OBC the own telemetries, housekeeping and status information.

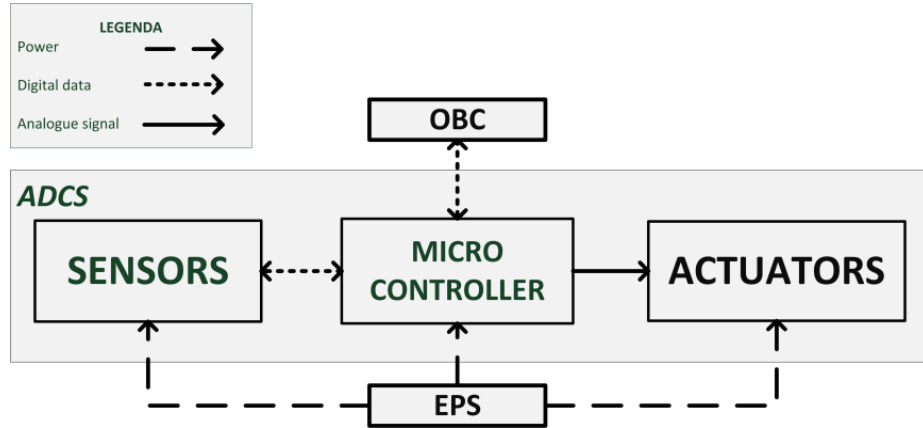


Figure 89: A-ADCS interfaces with the other subsystems

The A-ADCS receives the electrical power from the EPS and exchange information with the OBC. The components of the A-ADCS are selected through the functions/components matrix shown Table 34. The satellite angular velocities are measured by an IMU while three magnetometers measure the local EMF; the maneuver is guaranteed by three MT mounted orthogonally on the faces +X,-Y,+Z (of the body frame). All the “smart” operations and tasks are in charge of the microprocessor. The microprocessor has ARM architecture: this choice derives from the background and the experience already made in the past.

Function\equipment	Gyroscope	Magnetometer	Computer	PWM circuit	MT
To propagate the orbit			X		
To measure angular rate	X				
To handle the measurements data			X		
to measure the EMF		X			
to load models and look-up tables			X		
To define desired attitude			X		
To define desired angular velocities			X		
To evaluate the error between estimate angular velocities and desired angular velocities			X		
To evaluate the error between estimate attitude and desired attitude			X		
To compute the control torque values			X		
To execute the command				X	
To generate torques					X
To compare ADCS parameters			X		
To switch on/off ADCS hardware parts			X		
To insert/remove ADCS software tasks			X		
To check the commands			X		
To acquire health status data and power consumption			X		
To compare health status with thresholds			X		
To activate recovery mode			X		
To reconfigure ADCS HW parts			X		
To reconfigure ADCS SW tasks			X		
To acquire information			X		
To extract data			X		
To handle data			X		
To execute command			X		
To gather data			X		
To format data			X		
To send data			X		

Table 34: Matrix functions/components of A-ADCS

The blocks scheme of the A-ADCS is shown in Figure 90: black lines indicate the power connections, red lines indicate the digital data connection, and blue lines indicate analog signals. The ARM9 micro-controller passes data through a serial port to the OBC as well as it receives sensor measurements on a second serial port. The three magnetic torquers are commanded using a PWM logic circuit driven by the three micro-controller timer ports. ARM9 saves and load data and models from the internal memories (both volatile and non-volatile). MT are attached on three different connectors. All the power derives from the EPS thanks to the e-st@r bus: regulation on 3.3 V, 5 V and within the range [7, 8.2] V are needed.

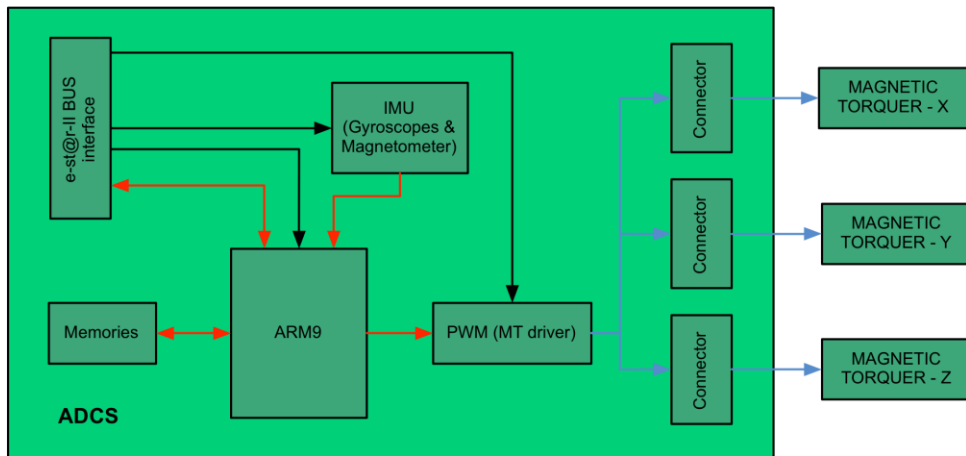


Figure 90: Blocks scheme of A-ADCS

4.4.2 M&S based design and verification of the A-ADCS of e-st@r CubeSat

The A-ADCSs of e-st@r CubeSats family have been designed and verified through the methodology proposed in this thesis. Table 35 summarizes the design and the verification activities carried out during the e-st@r program thanks to StarSim. For each stage in the life-cycle, the appropriate type of simulation has been chosen, and the objectives of each simulation are properly defined according to the phase of interest. Pay attention that the activity in the Operation phase is not available because e-st@r II has not reached the orbit yet.

4.4.2.1 Mode of operations and attitude determination and control algorithms

The components of the A-ADCS are chosen through simulations evaluating the accuracy of the determination and the control, the power consumption, the mass budget, and possible layout.

For e-st@r-I CubeSat, all the A-ADCS functions were switched on immediately after the release from the P-POD so a unique control mode was designed for nominal operations. In case any anomaly was detected (like low voltage of batteries or components failure), the A-ADCS was switched off. On e-st@r-II, three control modes have been defined. Taking into account that the A-ADCS is the payload, the choice is to activate it only upon command from ground according to the operations timeline. In this way, the commissioning phase of the mission shall be completed prior the ADCS is switched on, in order to verify the right working of the satellite and to avoid the discharging the battery in the power-expensive detumbling phase immediately after the release. This evaluation has been made thanks to the experience gained from the first satellite in which the ADCS was activated immediately after release.

	Feasibility	Design and development	Integration	Qualification/ Acceptance	Operations*
AIL	<ul style="list-style-type: none"> • To evaluate the feasibility of the ADCS design wrt the top-level, mission and system requirements • To check the orbit for coverage studies, mission profile definition and ADCS modes determination • To size the disturbances and evaluate their effects • To verify torques-free motion • To define the needed control torque(s) in any phase (granularity and peak values) • To predict critical control aspects 	<ul style="list-style-type: none"> • To analyze the possible control modes • To analyze the control laws and algorithms options in relation to the mission phase (detumbling, stabilization, maneuver) • To analyze the determination algorithms options • To evaluate alternative architectures (sensors, actuators) • To evaluate extrinsic and intrinsic performances “against high fidelity virtual model” • To verify the capabilities of controller and observer 	N/A	N/A	<ul style="list-style-type: none"> • To verify the operations through on orbit data • To predict future behaviors and potential failures • To validate models with data from orbit
SIL	N/A	<ul style="list-style-type: none"> • To support the software design and implementation: structure and functions • To verify the interface protocols between components (MT, IMU) • To size and choose the micro-controller 	<ul style="list-style-type: none"> • To design and verify protocols with OBC 	N/A	<ul style="list-style-type: none"> • To evaluate possible updates of software
CIL	N/A	<ul style="list-style-type: none"> • To define and verify the kernel • To verify the micro controller performance • To verify time management & synchronization 	<ul style="list-style-type: none"> • To verify the real hardware interfaces with OBC • To verify the interfaces with power systems 	N/A	N/A
HIL	N/A	<ul style="list-style-type: none"> • To support the design of the ADCS board • To verify the sensors and actuators performance • To calibrate sensors and actuators • To check the capabilities of the every circuit • To check the extrinsic performance 	<ul style="list-style-type: none"> • To verify the interface with all the other on board subsystems (both physical and logical) • To verify the “real”extrinsic performances 	<ul style="list-style-type: none"> • To verify functional and operational requirements for the ADCS (and, in general, for the satellite) after integration 	<ul style="list-style-type: none"> • To verify the behavior using on orbit data

Table 35: StarSim in the e-st@r life cycle

Moreover, the gradual activation of the on board hardware allows to manage the operations according to the available power. For these reasons the satellites modes are:

- ADCS - Mode 0: no A-ADCS component is switched on: it is the default mode.
- ADCS - Mode 1:
 - A. if the commissioning phase is successfully completed, ARM9 micro-processor starts its work and IMU and magnetometer are activated in order to evaluate the angular velocities from the telemetries;
 - B: at the end of the detumbling phase, the attitude determination using the measurements and the dedicated algorithms is performed.
- ADCS - Mode 2:
 - A. if all the telemetries parameters are in nominal way, the controller is activated so the satellite enters in the detumbling mode;
 - B. If the attitude estimation is completed, the stabilization mode is activate: the satellite maneuvers to reach the antenna pointing to nadir;
 - C. if a command with new desired pointing is required from GS, the satellite maneuvers to satisfy the request. It is important to notice soon that perform a maneuver with a 1U cubesat and only magnetic actuators results very challenging. In this sense, slew maneuver for e-st@r mission is the cherry on the top.

Going into the details of the A-ADCS subsystem, the mode of operation of A-ADCS can be decomposed in phases:

- *ADCS-dormant phase*. It is active when the satellite is in *Basic Mission* mode of operations. All the subsystem components are powered off.
- *Sensors phase*. After the satellite commissioning in when Dormant, Activation and Basic mission satellite modes are set, the IMU (as well as the microprocessor) is switched on mainly in order to acquire di measurements of angular velocities (given by the gyroscopes).
- *Detumbling phase*. After the release in orbit, the satellite motion is characterized by a high angular rate with respect to the inertial frame. Under these circumstances, it is really difficult to estimate the satellite attitude. In this situation, the controller aims especially at reducing the angular velocity, regardless of the attitude. For the control in the detumbling phase two solutions are taken into consideration:
 - The proportional (P) controller on the angular velocities measurements multiplied for EMF components;
 - The Bdot controller that is based only on the EMF measurements.

The detembling controller works until the estimated angular velocity of the body frames w.r.t. the inertial frame is at the same time less than 0.005 rad/s around each axis.

Performances of these solutions have been evaluated in terms of power consumption and time to reach the final desired condition.

- *Attitude acquisition phase*. When the satellite motion is sufficiently dumped, the attitude is estimated through the sensors measurements and ad hoc algorithms and software filters. From the navigation point of view, alternatives consist of:
 - “deterministic methods”: the options taken into account are q-triad, q-est and qmethod.
 - “recursive methods”: the options are Linear Kalman Filter (LKF), Extended Kalman Filter (EKF), and Unscented Kalman Filter (UKF) has been considered.

It should be taken into account that initial conditions on the state variables are not known when the A-ADCS is activated.

- *Stabilization phase*. Once the attitude has been estimated (during the acquisition phase), the satellite is almost stabilized around an undefined orientation. It has a low angular velocity but

its attitude is not the desired ones. To reach the required antenna pointing (nadir), it is necessary to align the body frame with the orbital frame (it is centered in the centre of the Earth, with the x-axis directed towards the direction of motion, the z-axis directed toward the centre of Earth and the y-axis completing the right hand system). Possible test control law solutions are:

- The PID controller is the simplest solution that allows reaching pointing accuracy lower and, in general, lower stability of the extrinsic performances. It results in a low computational cost.
- The LQR provides an optimal and more sophisticated solution. It starts from the estimated values of the state variables and stabilizes the satellite according to a functional cost that has to be minimized acting on weights on the variables state and the commands.

Performances has been evaluated in terms of time to acquire the final pointing, accuracy of the final pointing, power consumption, stability and robustness, capability to reach a new desired attitude.

- *Slew maneuver phase.* This mode foresees that a desired final attitude is setup and the A-ADCS works to satisfy this new pointing. For this mode only the PID controller has been evaluated because the accuracy requirement are more relaxed wrt the nadir pointing accuracy requirements.

Figure 91 shows the transitions among the various modes of the A-ADCS. Red lines highlight the transition commanded directly by OBC or from the GCS, black lines highlight the automatic transition, made when particular conditions arise, blue lines stay for special transitions that completely reset the system (i.e. hard reset or specific commands from the GCS).

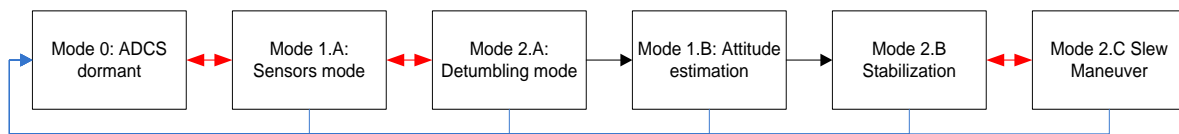


Figure 91: e-st@r-II ADCS operative modes

StarSim has been used to perform trade-offs analysis in AIL configuration.

The AIL final configuration of the simulator and the results from AIL simulations is obtained after many iterations of the methodology and now described hereafter. The subsystem's architecture (sensors, actuators and other devices) is frozen and the final choices for control modes and related controllers techniques, guidance strategies, and determination algorithms are implemented and verified together in the simulation unit. Moreover the mission profile is simulated and the subsystem performances are evaluated.

4.4.2.1.1 Basic AIL simulation during the feasibility phase

4.4.2.1.1.1 Objectives

The main objectives of this simulation session are the verification of the orbit, the EMF, the evaluation of the disturbance torques and the sizing of the required control torques.

4.4.2.1.1.2 Setup and configuration

The StarSim configuration for AIL simulations is shown in the next figures. Figure 92 shows the StarSim windows configuration for the file *setsimarch.txt*: only one process is built, no software or hardware interfaces are foreseen, i.e. the simulation runs exclusively on the simulation unit.



Figure 92: AIL simulations – processes settings

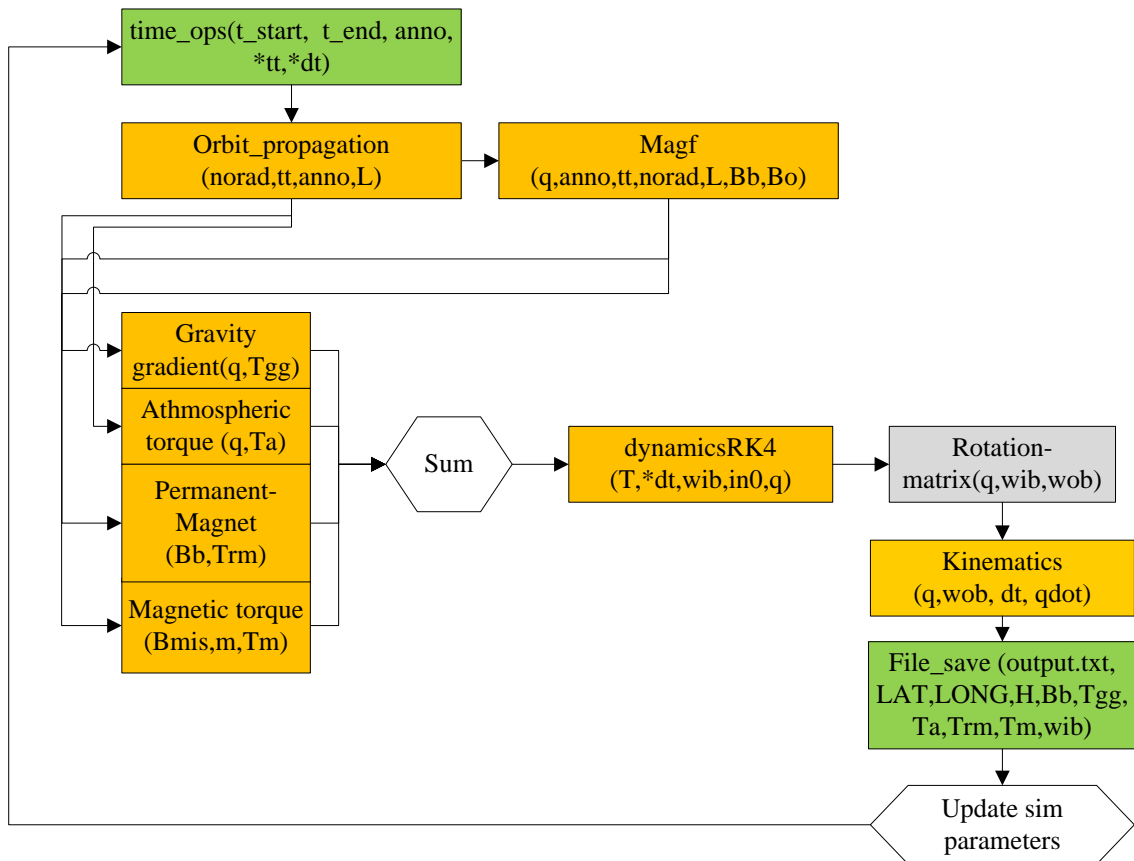


Figure 93: Basic AIL simulation – models flow

The *models flow* (in Figure 93) contains the functions for the time management, the orbit propagation (with J2 perturbation) model, the IGRF EMF model for the five-years period 2011-2015, the models of the four most influent torques acting on the satellite (residual magnetic dipole, gravity gradient, atmospheric drag, and the magnetic control torque), the dynamics and kinematics of a rigid body in the space, and the functions that permit to save data on the file selected by the user (output.txt). Each

model is setup accordingly the known information at this stage: the estimation made through CAD of the inertia matrix, the weight, the supposed launch date, the orbit parameters. Moreover, the simulation settings are: simulation time = 2 days, step-time = 0.5, no real time is required.

4.4.2.1.1.3 Results

The main results are listed hereafter:

- **Orbit propagation with J2 perturbation:** Figure 94 shows that the orbit propagator model properly simulates in time the orbit of the satellite. The on ground track line highlights the compliance of the “orbit propagation” models with the expected results.

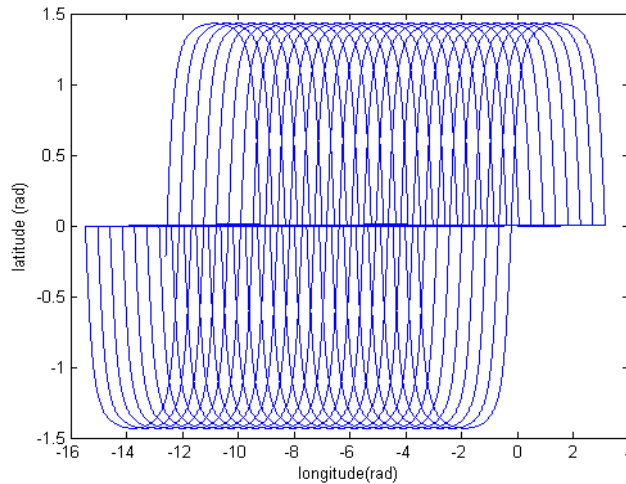


Figure 94: Basic AIL simulation - Orbit propagation

- **Earth Magnetic Field.** Figure 95 shows the EMF trend in the body frame: the three

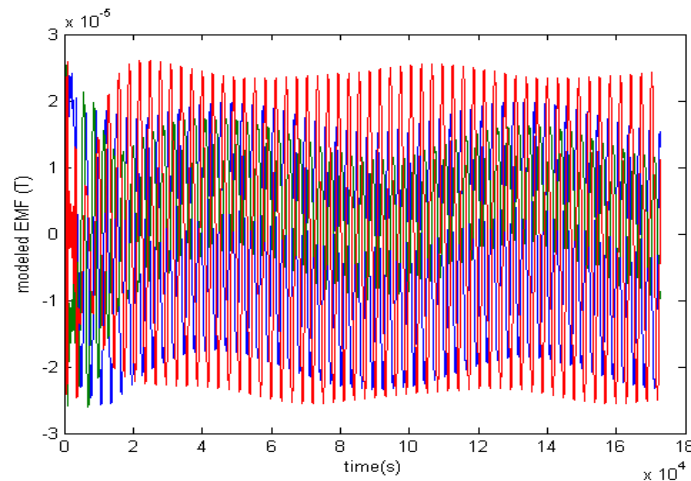


Figure 95: Basic AIL simulation - EMF generated from the model

- components (respectively red for wrt x, green wrt y and blue wrt y) of the EMF vector are highlighted. The obtained results are successfully compared with the official IGRF model (conveniently rotate with a 1-3-1 rotation []) output and no relevant differences arise.
- **Disturbance torques:** the peak values and trends of disturbance torques are evaluated, in Figure 96; the maxima (in modulus) of any considered torque are:
 - $\text{Max}(T_{gg}) = 1.0e-009 * [0.2312, 0.3201, 0.0889] \text{ Nm}$

- $\text{Max}(T_a) = 1.0\text{e-}013 * [0.2674, 0.2674, 0.2674] \text{ Nm}$
- $\text{Max}(T_{res}) = 1.0\text{e-}008 * [0.2676, 0.2673, 0] \text{ Nm}$

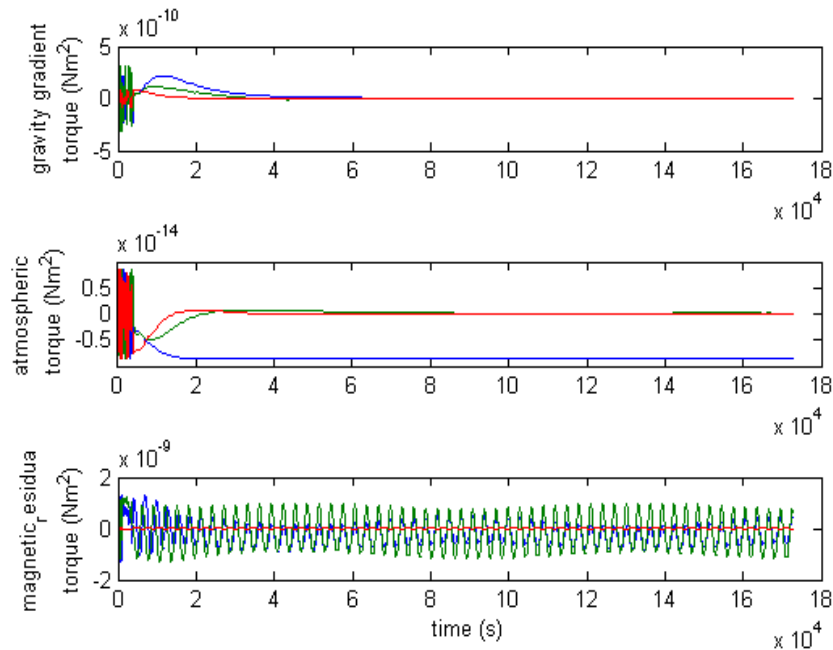


Figure 96: Basic AIL simulation - disturbance torques trends

The maximum values for the sum of the disturbance torques are: $1.0\text{e-}008 * [0.2880, 0.2980, 0.0089]$ compatible with the results expected [37].

About the trends:

- The atmospheric torques in steady state conditions mainly acting on two axes orthogonal to the orbit motion direction.[38]
- The magnetic residual torque has two main contributions around the 2 axes orthogonal to the dipole moment generated by the electronic boards lies in parallel plane wrt that formed by two axes. [39]
- The gravity gradient torque [1] present quite similar trend and values because the Inertia wrt the main satellite axis are quite similar ($I_x=0.001934$; $I_y=0.001879$; $I_z=0.001736$). $\text{Kg}\cdot\text{m}^2$
- **Dynamics and Kinematics models.** The attitude dynamics of the satellite has been evaluated thanks to a simulation in which no control torques are applied. Figure 97 shows the trend of the body angular velocities wrt the inertial frame. This confirms the correctness of the “dynamics” and “kinematics” models.

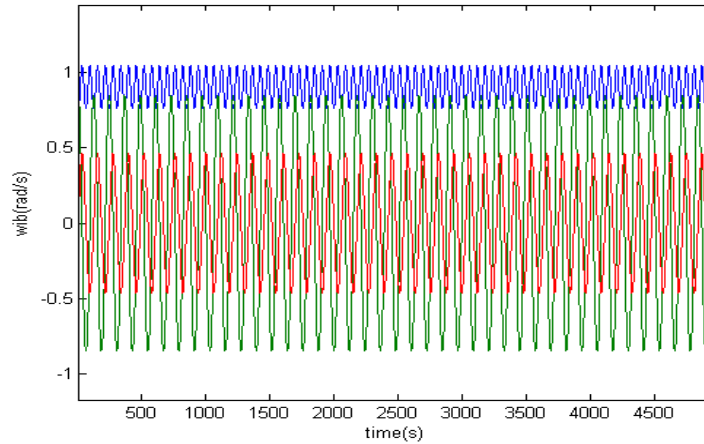


Figure 97: Basic AIL simulation –detail of the angular velocities when no control torques are applied

- The control torque sizing derives from the analysis of the disturbance torques and the torque-free motion acting together on the satellite. A large number of simulations has been run thanks to the StarSim’s capabilities in terms of speed of simulation and possibility of execution of more sessions in parallel. In this phase no optimization has been made to tune the control parameters, but the obtained results is sufficient to have a preliminary sizing.

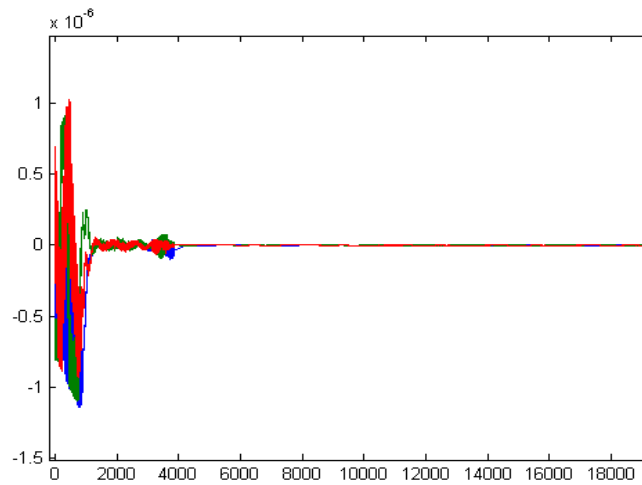


Figure 98: Basic AIL simulation -Control torques trend and sizing

- The maximum control torques of $2 \cdot 10^{-6}$ Nm is sufficient to damp the satellite after release and a minimum torque about 10^{-8} Nm guarantees a good pointing accuracy and an acceptable pointing stability for the e-st@r mission. Figure 98 shows the trend of the control torques.

4.4.2.1.2 Control techniques comparison via AIL simulation

4.4.2.1.2.1 Objectives

The simulation sessions aim at define the best control techniques to apply for the attitude control of the satellite taking into account the extrinsic performances, the required computational load, and the power consumption.

4.4.2.1.2.2 Setup and configurations

Only one process (the *simulation process*) shall be configured for this verification. See Figure 92 with the *setsimarch.txt* script for details.

Figure 99 illustrates the StarSim “models flow” for the actuators choice and control laws definition and verification; in particular, the figure shows the final configuration obtained at the end of the simulation sessions. Disturbance torques, orbit propagation, modeled EMF, dynamics and kinematics of the satellite with extra function (i.e. rotation calculation) are inserted as already shown above. The blue block contains the satellite control algorithms and laws, and the red blocks refer to actuators (3 MT) models and the control logic (PWM) algorithm.

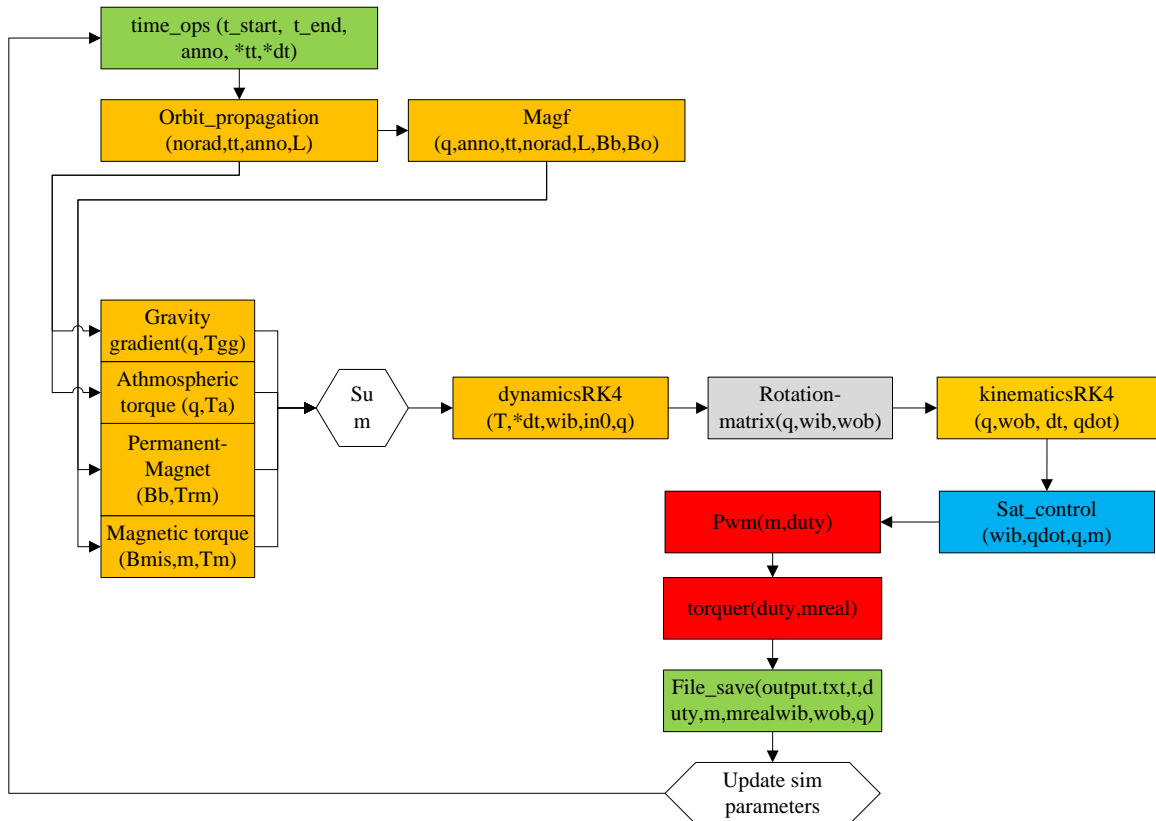


Figure 99: Control Techniques AIL simulation – models flow: actuators and control laws

All the developed models have been validated comparing the output with those obtained using MATLAB/Simulink models and simulating the same conditions. No evaluable differences arise from the comparison confirming that the simulator is able to rightly perform the planned simulation and reproduces the expected behavior of the virtual models.

4.4.2.1.2.3 Results

The considered actuators are magnetic torquers and reaction wheels. Thrusters were discarded at first because they are not easily applicable to CubeSat ADCS designs (at the present state-of-the-art technology). Reaction wheels resulted soon too bulky and, moreover, their power consumption was too high wrt to the power allocates for the ADCS: RW requires about 5 W of peak while the allocate power budget is 3 W. MT is the adopted solution because easiest and cheapest wrt any other. Moreover, the accuracy and granularity is sufficient for the program purposes and functional requirements. It depends on the PWM logic capabilities: it has been demonstrated through simulation

sessions that a granularity of $0.015 * 10^{-3} \text{ Am}^2$ is sufficient to reach the desired pointing as well as a $0.1 * \text{A m}^2$ is sufficient to guarantee damping of the satellite during the detumbling phase with an initial condition of about 1 rad/s for any axis.

The considered control laws for detumbling are the \dot{B} and $K * w \times B$. Figure 100 and Figure 101 reports the trend of the angular velocities (w) during the detumbling phase using \dot{B} ; Figure 102 and Figure 103 shows the trend of the same angular velocities using the control law $u = K * w \times B$. The figures highlight a better capability of \dot{B} method to damp all the three components rapidly; on the contrary $w \times B$ provides a better stability in steady state conditions. For this reason the second method has been adopted but, actually, no great differences are evident between the two methods because both provide proper solutions.

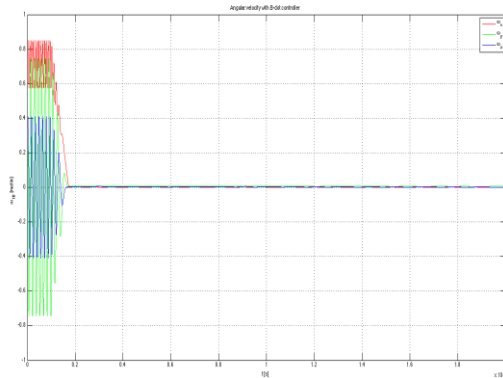


Figure 100: Control Techniques AIL simulation detumbling phase using \dot{B}

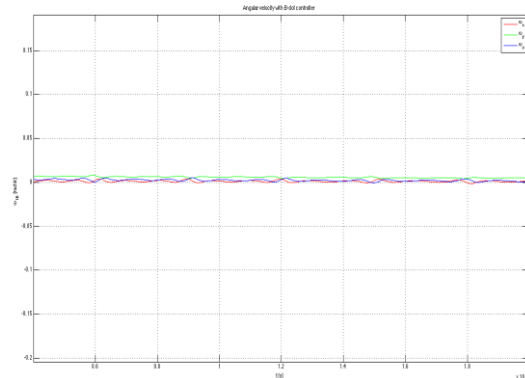


Figure 101: Control Techniques AIL simulation detumbling phase using \dot{B} (detail)

The considered control laws for the stabilization phase are the PID and the LQR techniques. A great number of simulations has been required to tune properly both the PID values and the LQR's Q and R matrices.

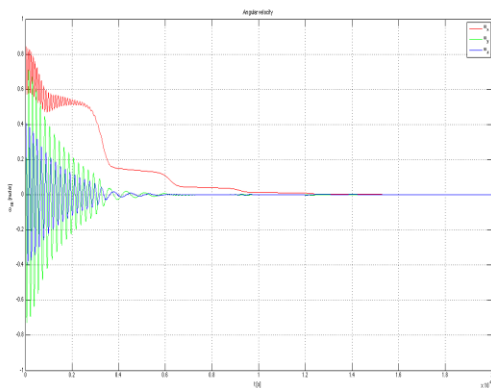


Figure 102: Control Techniques AIL simulation - detumbling phase using $w \times B$ control law

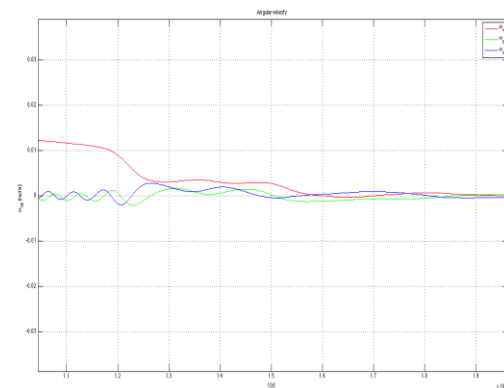


Figure 103: Control Techniques AIL simulation- detumbling phase with $w \times B$ control law (detail)

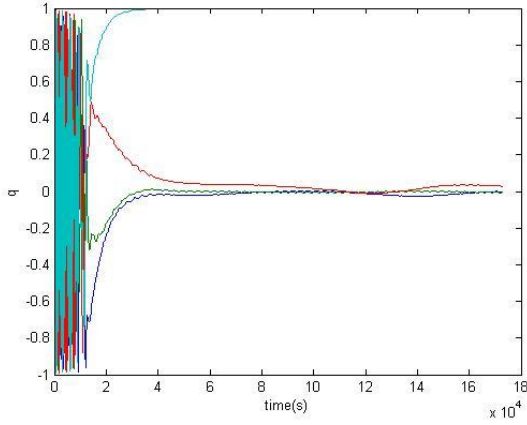


Figure 104: Control techniques AIL simulation - stabilization control, PID

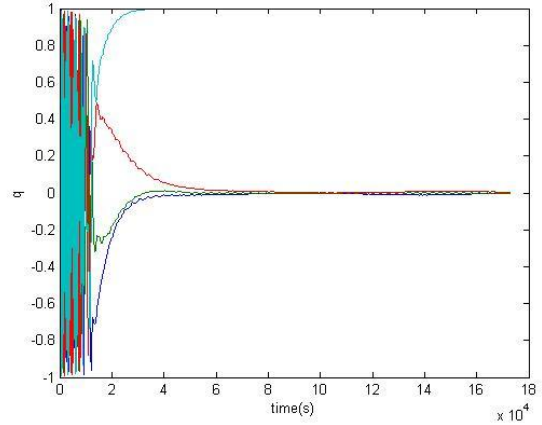


Figure 105: Control techniques AIL simulation - stabilization with LQR

The Figure 104 and Figure 105 reports the trends of the quaternion for both the solutions. As expected, LQR performs better than the PID when uncertainties are applied both in inputs and in outputs. However, a mix of the two control laws has been implemented: PD controller works in stabilization until a misalignment respect the nadir pointing major of 25° on each axis is measured, while within these range the LQR starts own operations because, as shown in Table 36, its performances are better than PID.

	PID			LQR		
	$x(^{\circ})$	$y(^{\circ})$	$z(^{\circ})$	$x(^{\circ})$	$y(^{\circ})$	$z(^{\circ})$
Pointing accuracy	[5 : 7]	[-1 : 1]	[-6 : 0]	[+0.5 : 1]	[-0.5 : 0]	[-1 : 0]
Stability	0.5°/10000 seconds	1°/10000 seconds	1.5°/10000 seconds	0.05°/10000 seconds	0.1°/ 10000 seconds	0.5°/ 10000 seconds

Table 36: PID vs. LQR performances

Moreover, it has been seen that the ADCS is able to detumble and point to nadir the satellite also after 7 days from the release without a decay of performances in terms of accuracy or power consumption. The Figure 106 shows that satellite is able to perform the detumbling maneuvers after a week of uncontrolled motion and starting from the initial condition: body angular velocities wrt inertial frame [1, 1, 0.8] rad/s. It is considered the worst release conditions of the satellite from the P-POD.

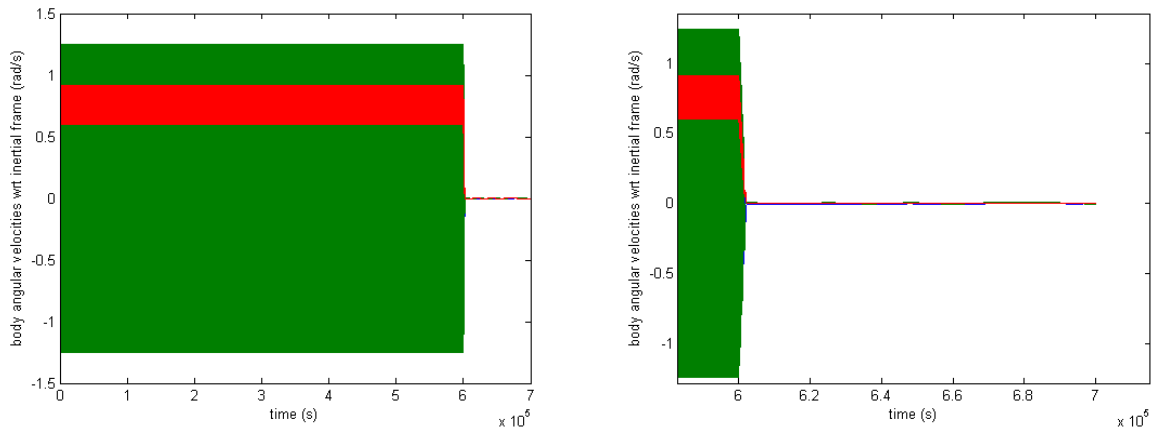


Figure 106: Free motion seven days simulation and successive detumbling

4.4.2.1.3 Determination algorithms comparison via ALL simulation

4.4.2.1.3.1 Objectives

The test aims at choosing and verifying the best solutions and attitude determination algorithms and sensors choice.

4.4.2.1.3.2 Setup and Configurations

Only one process (the *simulation process*) shall be configured for this verification. See Figure 92 with the *setsimarch.txt* script for details. Figure 107 shows the configuration of StarSim “models flow” used for the scope. Disturbance torques, orbit propagation, modeled EMF, dynamics and kinematics of the satellite with extra function (i.e. rotation calculation) are inserted, as already described above. The figure refers to the final configuration in which the red blocks contain the models of the adopted sensors (3 axial magnetometer and 3 gyroscopes). The blue blocks contain the determination algorithms: the final choice foresees q-methods starting from the magnetometer measurements and the EMF modeled values (with respect the current orbit position) and EKF on the angular velocities measurements and the entire process. The final choice has made on the application of q-method on two successive measurements of EMF and the EKF on the the state variables of the system. In particular, using q-method, the estimation of the initial attitude is done in order to initialize the integrator within the models and the KF. EKF is the effective state estimator. The tuning of its values has been a quite complex process because it has been designed considering white, Gaussian noises but its convergence has been verified taking into account colored noise and no idealities as temperature, scale factor, misalignment, bias instability. In this way the higher difficulty is translated in a higher confidence in the filter because a more robust solution has been finally reached.

The simulation time is 2 days, step time is equal to 0.5 seconds, and the real time option is not selected.

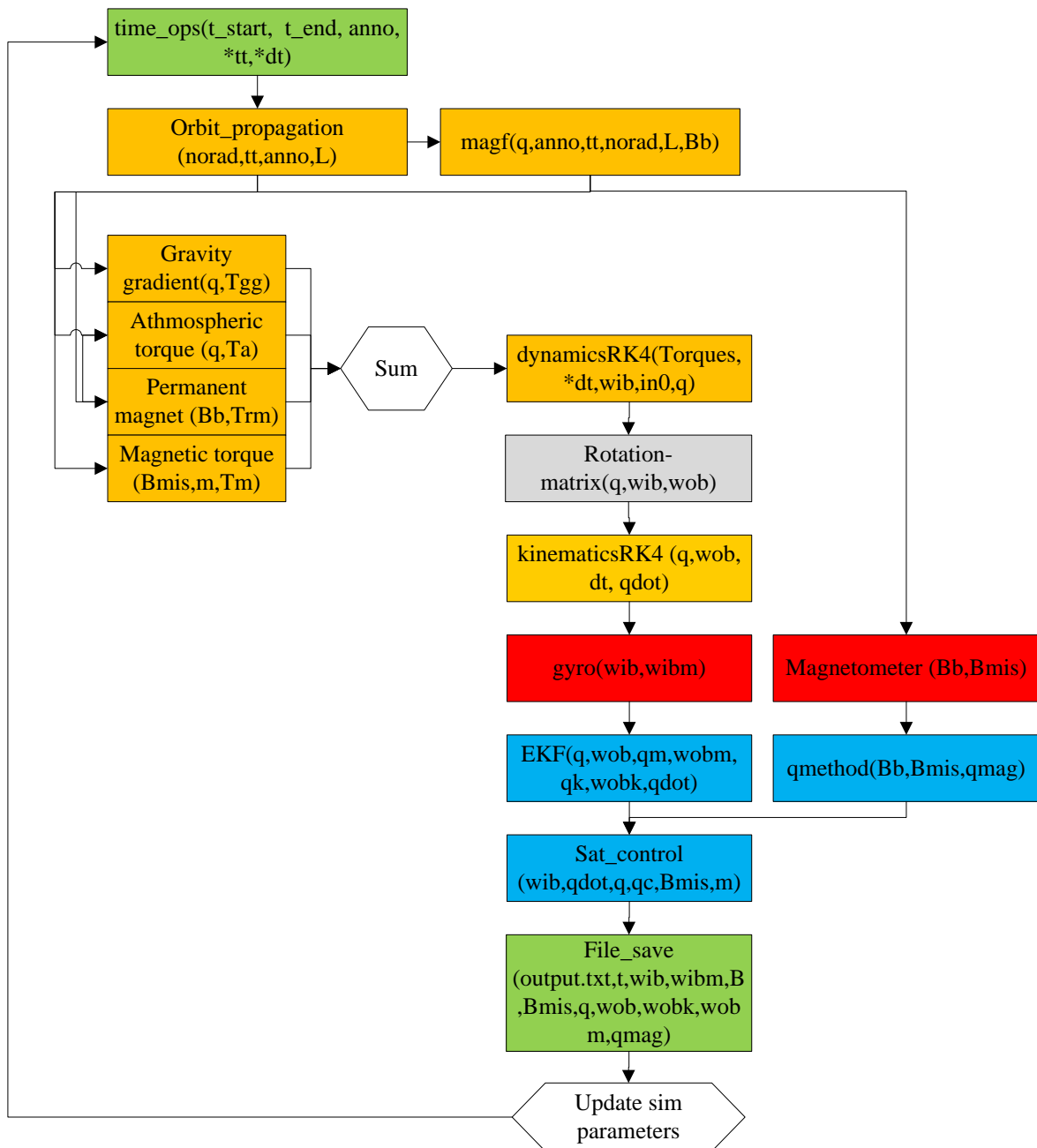


Figure 107: Determination algorithms comparison via AIL simulation – models flow

4.4.2.1.3.3 Results

1. The sensing device is an IMU containing a 3-axial gyroscope and 3 mono-axial magnetometers. The sensor is also constituted by three accelerometer not used for determination attitude but only for calibration. One of the early hypotheses about sensors for determination was constituted by the IMU and by the solar panels used as coarse Sun sensor. This was the configuration for e-st@r-I that has been substituted because the solar panels telemetry acquisition sometimes resulted to slow and the risk is to have old values when the determination algorithms performed. Moreover, a traditional sun sensor drawback has been taken into account: the impossibility to have the measurements during the eclipse periods. The IMU chosen has the merit of self manipulate the acquisitions of its internal sensors and correct them accordingly: in particular, it just compensates the noise generated by not-linearities, bias instability and temperature. Uncompensated noises remain but to really evaluate their effect it

should be made after tests in a thermal chamber but the impossibility to perform this kind of test leads to apply software correction adding randomly an offset to the values that will be filtering by the EKF on the plant.

- The considered algorithms for determination are the Q-est and Q-method for deterministic/statistical methods and EKF for the recursive methods. Q-est/Q-methods are applied to two consecutive measurements and the correlative IGRF model values of the magnetometer and EKF on the “plant” and the gyroscope measurements. After the detumbling phase (made using the gyroscope measurements), the attitude acquisition is made with Q-method because it has higher performances wrt the Q-est :

- $MKE(q_{est})=[0.003723 \ 0.002801 \ -0.002567 \ -0.0002946]$,
- $MKE(q_{method})= [0.003128 \ 0.001916 \ -0.0017253 \ 0.0002602]$

Moreover, its implementation is not too difficult and the request of computational resources is compatible and not so demanding modern micro-processors.

Actually also q-TRIAD were evaluated but it provided results inefficient in terms of accuracy wrt the requirements satisfaction.

Figure 108 shows the trends of the modeled EMF and the measured ones (obtained from the magnetometer model).

Their values are used by the qmethod and qest algorithms to determine the attitude during the acquisition phase. Moreover, the $MKE(B)$ has been computed:

- $B-Bm=[0.021297458645963 \ 0.030170592688397 \ -0.137637367679484]*e-8$ Tesla

Q-method accuracy using the magnetometer measurements results less than the values computed through EKF applied on gyro measurements and the satellite virtual model, the satellite dynamics and kinematics with its uncertainties and disturbances:

- $MKE(q_{EKF}) = [0.000082 \ 0.0003765 \ 0.000771 \ 0.001115]$
- $MKE(q_{method})= [0.003128 \ 0.001916 \ -0.0017253 \ 0.0002602]$

For these reasons, in nominal conditions, the attitude determination is committed to the outputs of the EKF periodically corrected by the q-method outputs

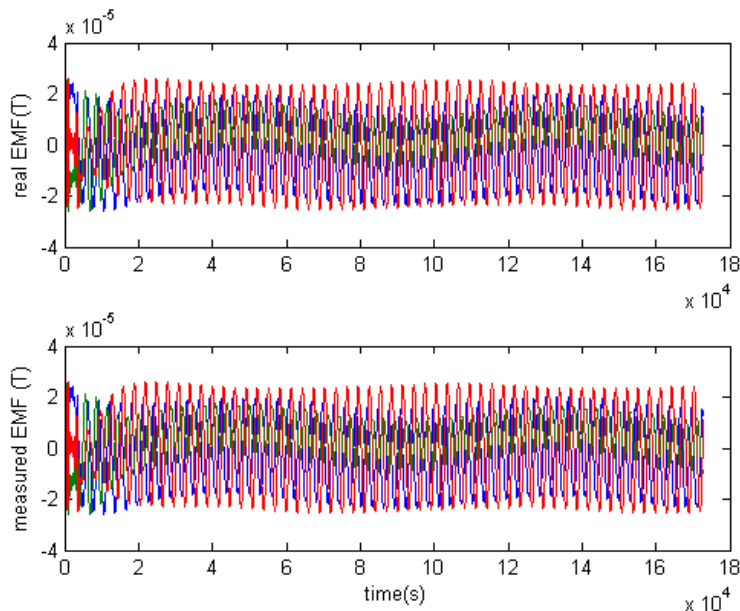


Figure 108: Determination algorithms comparison via AIL simulation –modeled and measured EMF

3. Figure 109 reports the trend (in the first 5000 seconds of simulation, for a good reading of the plot) of the simulated and measured (from the gyroscopes model) angular velocities of the body frame wrt the orbit frame. Both are used by the Extended Kalman Filter to estimate the attitude (qEKF). Moreover, the MKE(wob) has been computed and is equal to $[-0.0564 - 0.064 - 0.0577] * 10e-5$ rad/s.

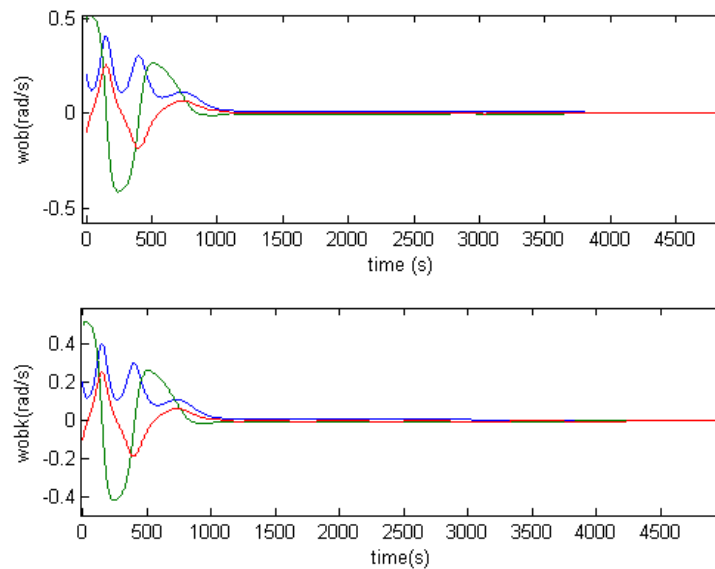


Figure 109: Determination algorithms comparison via AIL simulation – simulated and measured angular velocities (detail – first 5000 seconds)

4.4.2.1.4 Complete AIL simulation for final performance analysis

4.4.2.1.4.1 Objectives

The test objectives are the evaluation of the extrinsic performances and the validation of the ADCS design with an as high as possible fidelity of the virtual models.

4.4.2.1.4.2 Setup and configurations

Only one process (the simulation process) shall be configured for this verification. See Figure 92 with the *setsimarch.txt* script for details.

The final “models flow” for complete AIL simulations session is illustrated in Figure 110; it is the highest fidelity virtual models developed within the e-st@r program and it is defined as follows:

- Orbit propagation with J2 perturbation and aerodynamic disturbance forces
- EMF (IGRF2011-2015)
- Disturbance torques:
 - Atmospheric torques
 - Gravity gradient
 - Magnetic residual due to electronic devices
- Dynamics and Kinematics of the satellite with Runge-Kutta methods for the integrator computation.
- Sensors: MT9 IMU models (with magnetometers and gyroscopes). The gyroscopes main parameters are: scale factor matrix= $[0.269 -0.0005 0.0011; 0.0001 0.2586 -0.0231; -0.0011 -0.0005 0.2639]$, bias_digital = $[32582 32082 34490]$, imu_accuracy= $1.59762 * 0.0001$. The magnetometers parameters are: scale factor matrix= $[9715 0$

0;0 8584 0; 0 0 7932], Magnet sensitivity rotation matrix = [1.000 -0.007 -0.006; -0.060 0.988 -0.009; -0.051 0.010 0.999], Misalignment =[1.0000 0.0035 0.0000; -0.0035 1.0000 0.0017; 0.0000 -0.0017 1.0000]. bias =[32400; 32568; 32698]; noise vector =[0.5-rand;0.5-rand;0.5-rand];

- Actuators: 3 MT (the parameters are: coils number= 105, resistance = 49 ohm, V=5 V, mean area = $6100 \cdot 10^{-6} \text{ m}^2$, copper resistivity = $1.78 \cdot 10^{-8} \text{ ohm} \cdot \text{m}$)
- Control law: proportional on the angular velocity of the body wrt inertial frame for the detumbling, PID and LQR on the angular velocity of the body wrt orbital frame and the attitude (expressed with quaternion) for the stabilization phase, and the PID on the same variables of the stabilization phase to perform slew maneuvers.
- Control logic for the actuators: the PWM logic algorithms; set parameters are: PWM sample for second = 2000, Max voltage = 5Volt.
- Attitude determination algorithms: q-methods deterministic algorithms starting from the magnetometer measurements and EMF models, and EKF on the gyroscopes measurements and on the state variables of the satellite representation.

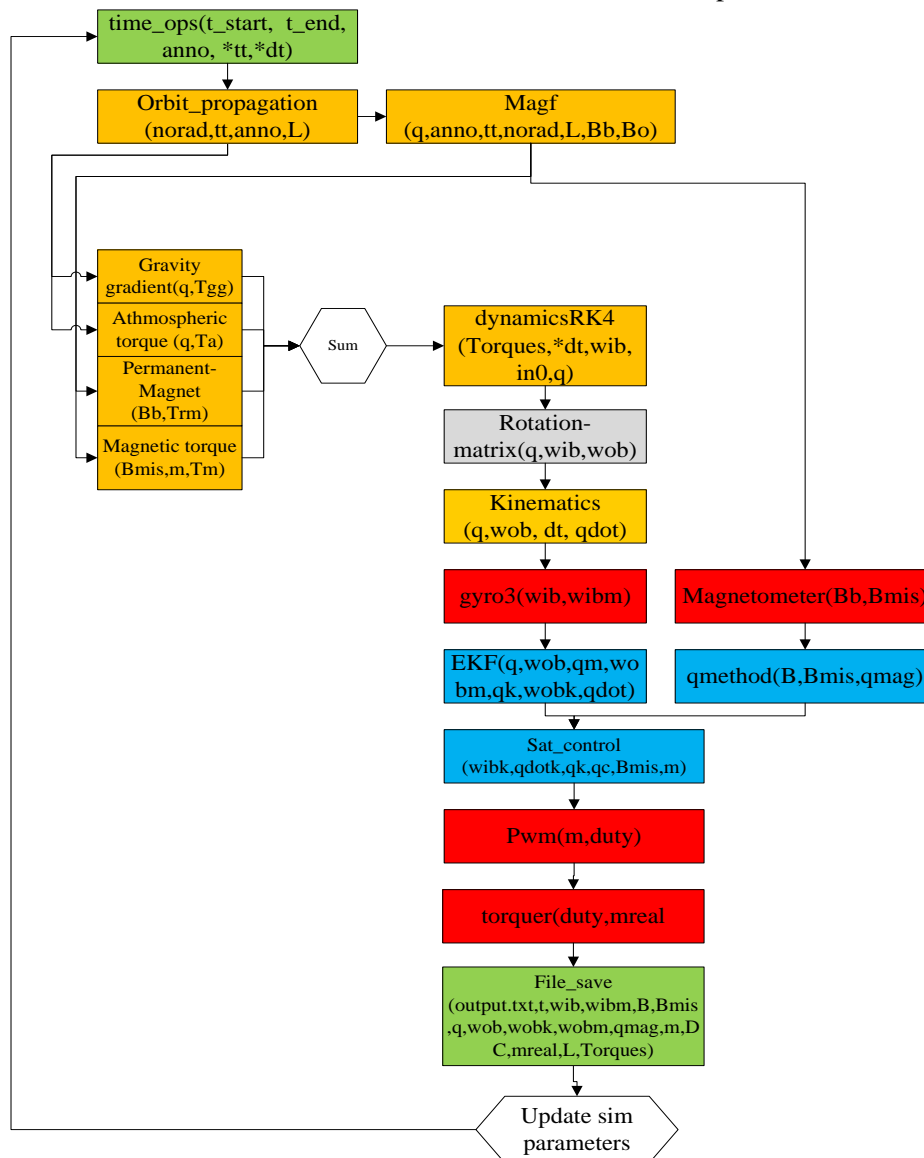


Figure 110: Complete AIL simulation – models flow

Moreover, algorithms and functions to manage time and saving simulation data (satellite angular velocities, attitude, measurements of the sensors, actuators consumption are selected. The mission profile foresees that for 10000 seconds the ADCS is switched off, then the detumbling phase starts. When this phase is finished, the attitude acquisition mode is activated and the attitude is estimated before with q-method, after 5000 seconds, from the EKF and the stabilization mode starts to work. The simulations last 2 days (172200 seconds) and non real time settings are required. The step time is 0.5 seconds (according to the foreseen control frequency = 2 Hz).

```

AIL_complete_estarII.c
Name : Process.c

#include "headers.h"
#include "function_header.h"
#include "port_libs.h"

int main(int argc, char* argv[]) {

    /* *****
     *
     *              INITIALIZATION
     *
     * ***** */

    struct pipe *pipe_ptr;
    struct serial *serial_ptr;
    int n_pipes = 0, n_serials = 0, i = 0;

    if (argc < 2) {
        printf("Wrong number of arguments passed.");
        exit(EXIT_FAILURE);
    }

    i = 0;

    /* *****
     *
     *              VAR POOL
     *
     * ***** */

    struct timeval end, start;
    double wob[3] = { 0.6, 0.6, 0.05 };
    double wib[3] = { 0.6, 0.6, 0.05 }, norad[8] = { 98, 0, 0, 0, 0, 0.0010, 0,
    0 }, tt = 0, anno = 0, m1[3] = { 0, 0, 0 }, B[3] = { 0.00001,
    0.00001, 0.00001 }, I[6] = { 0 }, L[3] = { 0, 0, 600000 }, T[6] = {
    20, 20, 20, 20, 20, 20 }, dt = 0.5, m[3] = { 0, 0, 0 }, q[4] = { 0,
    0, 0, 1 }, qc[4] = { 0, 0, 0, 1 }, wibm[3] = { 0.6, 0.6, 0.05 },
    wobm[3] = { 0.6, 0.6, 0.05 }, in0[3] = { 0 }, qdot[4] = { 0 },
    qm[4] = { 0, 0, 0, 1 }, qk[4] = { 0, 0, 0, 1 }, qdotk[4] = { 0 },
    wobk[3] = { 0.2, 0.2, 0.2 }, wibk[3] = { 0.6, 0.6, 0.05 },
    qdotm[4] = { 0 }, Torques[3] = { 0 }, Tgg[3] = { 0 }, Ta[3] = { 0 },
    Trm[3] = { 0 }, Tm[3] = { 0 }, Bmis[3] = { 0 }, qmag[4] = { 0 };
    float n_coils = 176, tf[6] = { 0 }, m_max = 0.122, area_coil = 0.0048;
    int DCs[3] = { 500, 500, 500 }, contatempo = 0;
    FILE * tp;
    double anno0, day, hour, mins, sec, tt1;
    double daysmo[12] =
        { -1, 30, 58, 89, 119, 150, 180, 211, 242, 272, 303, 333 };
    int mese;

    system("rm -r -f /home/lawrence/Dropbox/git/Simulatore/e-st@rII_test/Debug/LogFiles/");
    system("mkdir /home/lawrence/Dropbox/git/Simulatore/e-st@rII_test/Debug/LogFiles/");

    /* *****

```

Figure 111: Complete AIL simulation: C++ code skeleton for the simulation process in complete AIL simulations (I)

Figure 111 shows the C++ code initialization lines of the *simulation process*. Three libraries are included and they refers to the StarSim databases and C++ .header files libraries.

No interface among processes and/or real hardware is requested so the relative variables are set to 0. All the model variables are defined and initialized according to the choices of the user and the type: angular velocities (wib,wob) and quaternion (q), the orbit parameters (in terms of TLE), latitude, longitude and height (L), desired states (q_c), sensors settings, EKF settings, torques, magnetic torques parameters (n_coil, m_max, area_coil), torques (all the T), etc...

The last two lines serve to clean the log folder and create a new instance containing the new log files according to the user choice.

```

while (1) {
    gettimeofday(&start, NULL);

    time_ops(start, end, anno, tstart, &tt, &dt, tf);
    orbit_propagation(norad, anno, tt, L);
    magf(q, anno, tt, norad, B, L);
    gravity_gradient(q, Tgg);
    atmospheric_torque(q, Ta);
    permanent_magnet(B, Trm);
    magnetic_torque(m, Bmis, Tm);
    sum(Tgg, Ta, Trm, Tm, Torques);
    dynamics_RK4(Torques, dt, wib, in0, q);
    Rotation_matrix(wib, q, wob);
    kinematics(q, wob, dt, qdot);
    if (tt - tt1 > 5400) {
        gyro(wob, wobm);
        magnetometer(B, Bmis);
        kinematics(qm, wobm, dt, qdotm);
        qmethod(B, Bmis, qmag);
        EKF(q, wob, qm, wobm, qk, wobk);
        wob_to_wib(wobk, qk, wibk);
        kinematics(qk, wobk, dt, qdotk);
        if (tt - tt1 > 10800) {
            controllosat(wibk, qk, qdotk, qc, Bmis, m);
            assign_ms_to_m1(m1, m);
            pwm(m, DCS, n_coils, area_coil, m_max);
            torquers(m_max, n_coils, area_coil, m, DCS);
            file_save(&contatempo, wib, wibm, wob, qk, wobk, qmag, B, Bmis, tt, L, m, DCS, tf, &tt1, Torques);}
        }
        file_save(&contatempo, wib, wibm, wob, qk, wobk, qmag, B, Bmis, tt,
                L, m, DCS, tf, &tt1, Torques);
    }
    gettimeofday(&end, NULL);
}

```

Figure 112: Complete AIL simulation – C++ code skeleton for the simulation process in complete AIL simulations (II)

Figure 112 highlights the loop: *gettimeofday* is the function that manages the UTC time, *timeops* manages the simulation time, *orbit_propagation* contains the orbit propagator model. *Magf* has the lookup tables and the functions related to IGRF model. The chosen torques models are represented by *gravity_gradient*, *atmospheric_torque*, *permanent magnet*, *magnetic torque* and their outputs are summed and passed to dynamics and kinematics models (containing RungeKutta integrators). “If” conditions derive from the definition of the mission profile: when the simulation time is higher than 5400 seconds the determination functions are activated as well as, after 10800 seconds, the controllers functions.

Going into the details, gyroscopes and a tri-axial magnetometer models simulate the behaviour of the sensors; from their outputs q-method and EKF algorithms are applied obtaining the state variables estimation. *Controllosat* function implements the control laws and strategies, *pwm* functions

reproduces the behaviour of the PWM logic and *torquer* simulates the outputs of the real actuators. *File_save* allows to save all the interesting variables values, as selected by the user.

4.4.2.1.4.3 Results

Apart for the results about the determination and the control laws already presented, the Figure 113 reports the attitude of the satellite (expressed in terms of Euler angles).

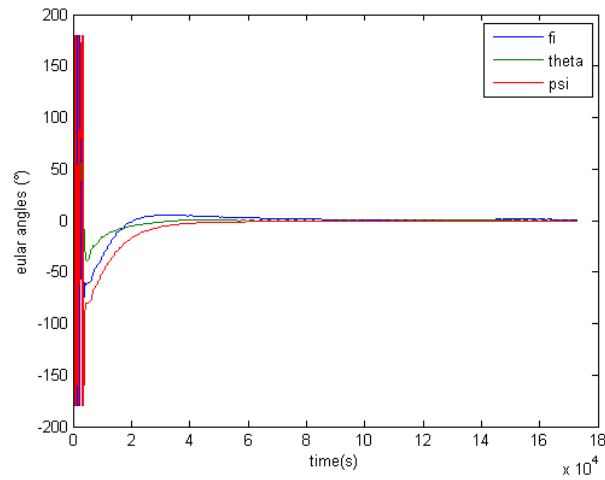


Figure 113: Complete AIL simulation - satellite attitude (Euler angles)

The desired attitude (the nadir pointing) is reached with an accuracy of about 1 degree for each axis and a pointing stability of $0.1^\circ/10000$ sec wrt to any axis. The accuracy of 5° is reached within the interval 18000-36000 seconds after the start of the stabilization phase.

Figure 114 shows the attitude determination made through the q-method algorithms and the EKF. The main results are already discussed previously.

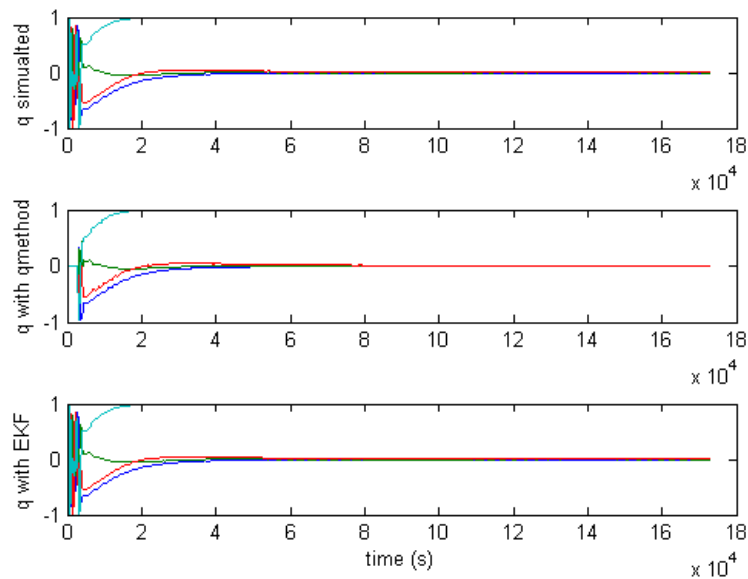


Figure 114: Complete AIL simulation - estimated quaternion using the magnetometer measurement (q-method) and the gyroscope measurement (EKF)

Figure 115 reports the commands (the dipole moment generated by the MT) computed using the control laws chosen. In particular, the trend in the mostly power expensive detumbling phase is zoomed. The total power consumption is about 750 mW in the first three hours.

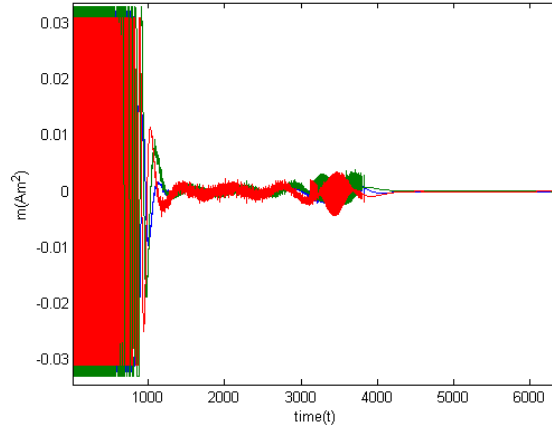


Figure 115: Complete AIL simulation - dipole moment (m), generating control torque, trend

4.4.2.1.4.4 Intrinsic performances

The study of the intrinsic performance on the uncontrolled and controlled system has been carried out. In particular, the stability analyses, the definition of the stability margins, the robustness and the robust performance analyses have been studied. The first step is the linearization of dynamics and kinematics represented by state variables equations:

$$\begin{aligned}\dot{x}(t) &= Ax(t) + B(t)u(t) \\ y(t) &= Cx(t)\end{aligned}$$

where $x=[q_1 \ q_2 \ q_3 \ \omega_x \ \omega_y \ \omega_z]$ is the state variables vector, $u(t)=[m_1 \ m_2 \ m_3]$ is the controlled inputs vector and $y(t)=x(t)$ is the outputs vector of interest. A, B, C are the matrices:

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ -4\omega_c^2\sigma_x & 0 & 0 & 0 & 0 & \omega_c(1-\sigma_x) \\ 0 & -3\omega_c^2\sigma_y & 0 & 0 & 0 & 0 \\ 0 & 0 & -\omega_c^2\sigma_z & \omega_c(1+\sigma_z) & 0 & 0 \end{bmatrix} \text{ is the state matrix}$$

Where $\omega_c=0.001$, $\sigma_x = (I_y-I_z)/I_x$, $\sigma_y = (I_z-I_x)/I_y$; $\sigma_z = (I_x-I_y)/I_z$ and

$$I = \begin{bmatrix} I_x & 0 & 0 \\ 0 & I_y & 0 \\ 0 & 0 & I_z \end{bmatrix} = \begin{bmatrix} 0.0017 & 0 & 0 \\ 0 & 0.0018 & 0 \\ 0 & 0 & 0.0019 \end{bmatrix} \text{ is the Inertia matrix.}$$

$$B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & B_z & -B_y \\ -B_z & 0 & B_x \\ B_y & -B_x & 0 \end{bmatrix} \text{ is the inputs matrix}$$

where B_x , B_y , B_z are the time varying components of the EMF in the body axes. For the analysis, mean values has been considered.

$$C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \text{ is the outputs matrix.}$$

These matrices represent the virtual model of the satellite obtained linearizing the non linear equation of dynamics and kinematics for the equilibrium point, $x=[0 \ 0 \ 0 \ 0 \ 0 \ 0]$.

- The stability analysis on the uncontrolled system is carried out calculating the eigenvalues of the matrix A. The six results are:

$$\lambda_1 = -0.0 + 83.51127i$$

$$\lambda_2 = -0.0 - 83.51127i$$

$$\lambda_3 = 0.0 + 13.32550i$$

$$\lambda_4 = 0.0 - 13.32550i$$

$$\lambda_5 = 0 + 57.735026i$$

$$\lambda_6 = 0 - 57.735026i$$

Four out of the six eigenvalues have positive real part: it means that the uncontrolled system is unstable.

- Controllability: the rank of the controllability matrix is 6 so the system is completely controllable from the inputs
- Observability: the rank of the observability matrix is 6 so the system is completely observable from the outputs.
- Stability of the controlled system made using the Lyapunov theory. The controlled system results asymptotically stable for a certain number of K. In fact, the eigenvalues are:
 - $\lambda_1 = -0.00264 + 0.00294i$
 - $\lambda_2 = -0.00264 - 0.002946i$
 - $\lambda_3 = -0.00265 + 0.0026i$
 - $\lambda_4 = -0.00265 - 0.0026i$
 - $\lambda_5 = -0.00116$
 - $\lambda_6 = -0.00021$

4.4.2.1.4.5 Extrinsic performances

The extrinsic performances measured after the attitude acquisition phase:

- RKE(q)= [0.00361 0.00245 -0.0034 -0.00153] NB. wrt the last simulation acquisition
- APE (q)= [0.001125 0.00088 0.007 0.00127]

- $MPE(q) = [0.028 \ 0.0042 \ -0.007 \ 0.00357]$
- $RPE(q) = [0.0269 \ 0.00367 \ 0.00082 \ 0.0035]$ NB. wrt the last simulation acquisition

Moreover, it constitutes the basis for all next configurations (SIL, CIL, and HIL): in fact, the virtual model holds even in later stages apart from the parts that become physical models along the product life cycle (embedded systems and equipment). This highlights another advantage of the SimStar as unique platform for any design and verification phase: the reusability of models (and, how shown in the next paragraphs, interface and items) among the different configurations.

4.4.2.2 SW design and test

The complete functional flow diagram of the final software for e-st@r is shown in Figure 116.

The ADCS software is structured as follows:

- “*initialization*”: it foresees that all the needed settings are made: I/O physical ports and software interfaces, models constant and variables are initialized
- “*interface loop*”: it is an infinite loop in which the RD129 acquires sensors and status data and exchanges telemetries and commands with the OBC. Within the loop:
 - the time is managed, maintaining both the mission and the UTC time
 - the string with the OBC information is received, read and the data are extracted from the raw.
 - NORAD parameters and desired attitude are updated (if any) according to the received commands and the all the telemetry data from OBC are updated.
 - the operative mode is individuated. If the *ADCS-MODE0* is active the execution flow passes to format and send the ADCS telemetry data to OBC and saves all the information into files (protection against the reboot). If *ADCS-MODE1* is active, the determination loop is active, or if *ADCS-MODE2* is active, also the control loop is active.
- “*determination modes*”: they are activated when the angular velocities only and the attitude and angular velocities of the satellite shall determine using the chosen algorithms: q-method and EKF.
- “*control modes*”: they are activated when the new commands are computed (according to the required control laws) and the PWM values are set for the driving circuits.

All the implemented functions are developed and tested both step by step and integrated.

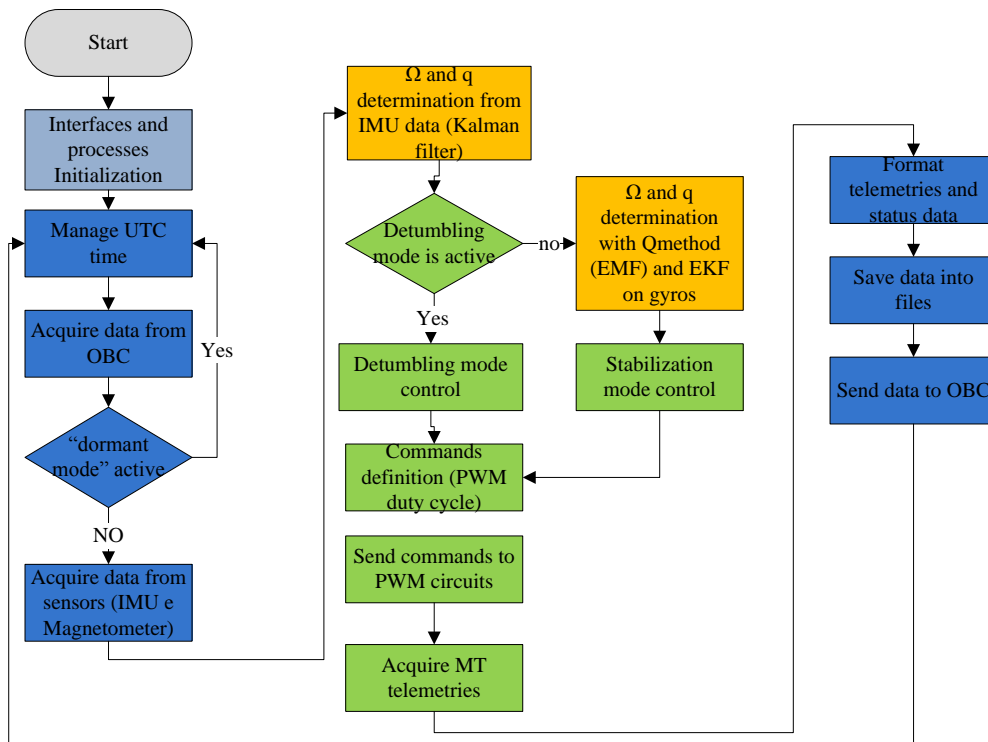


Figure 116: main ADCS software flow chart

4.4.2.2.1 SIL simulation for IMU verification:

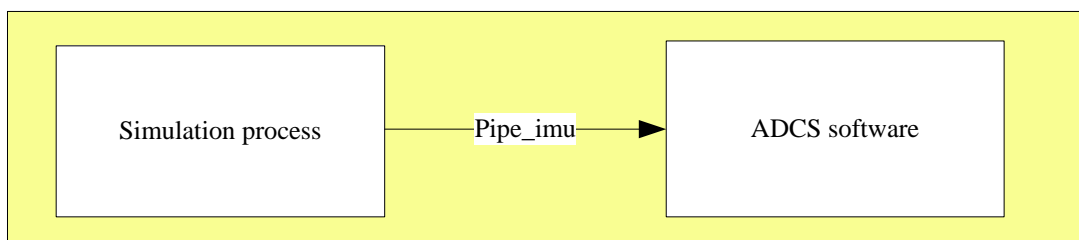
4.4.2.2.1.1 Objectives

The test aims at verify the ADCS software capability to extract data from a formatted IMU packet.

4.4.2.2.1.2 Setup and configurations

SIL configuration allows the verification of communication protocols between two different processes.

Figure 117 shows the SIL configuration for the verification of the IMU protocol and the ADCS software’s capability to acquire and manage the measurements.



```
setsimarch_IMU_SIL.txt x
####
1
tempistica 1
priority 1
path /home/ws1/Dropbox/git/Simulatore/Sim/SIL_IMU/Debug/SIL_IMU
pipe pipe_imu /home/ws1/Dropbox/git/Simulatore/Ports/Pipes/ w

####
2
tempistica 1
priority 1
path /home/ws1/Dropbox/git/Simulatore/ADCS/SIL_IMU/Debug/SIL_IMU
pipe pipe_imu /home/ws1/Dropbox/git/Simulatore/Ports/Pipes/ r
```

Figure 117: SIL simulation for IMU verification: processes settings

Two processes are involved: one (*simulation process*) generates the formatted IMU outputs. It computes the angular velocities values, the EMF, the linear accelerations and formats these data according to the custom protocol of the IMU, emulating the functions in charge of the sensor’s micro-processor. The formatted string is passed to the second process (called *ADCS_software*), with a pipe (labeled *pipe_imu*), where runs the piece of executable code devoted to validate packets and extract data (angular velocities, accelerations and EMF components).

Figure 118 shows the “models flow” of the two processes. The *simulation process* (on the left) simulates the system motion and the on orbit environment (orange blocks), the gyroscope and magnetometers behavior (red blocks), the data and signal handling, data saving and time management (green blocks). In particular, there are the model of the ADC need to effectively replicate the sensor behavior, the special function built to simulate the data formatting and packing made by the processor of the sensor. The LINUX generic function *write* allows sending via pipe the IMU-string to the *ADCS software*. Another solution can use a special function generating IMU strings. In both cases a properly formatted string is passed to the *ADCS software* process. This process acquires string thanks to the generic LINUX function *read*, validates each string and extracts values; moreover it saves on file (*imu_adcs.txt*) and visualizes information. The post-processing activities consist of reading the two binary files *IMU_sim.txt* and *IMU_ADCS.txt*, saved independently by the two processes and comparing both the binary sequences and the real “physical” data.

These simulation parameters have been chosen:

- Simulation time = 2 minutes/30 minutes/1 hour/4 hours
- Step time= 0.5 seconds
- RT is activated

4.4.2.2.1.3 Results

Different simulation sessions are performed. The obtained results show that the software working on the *ADCS software* is able to acquire and reconstruct the IMU information from the flow of bits received in input. The software extracts data from the formatted packets transmitted by the simulated IMU every 500 ms for half an hour (8999 packets), one hour (17993 packets) and for hours (71984 packets) and no relevant errors occur during the operations of acquisition and reconstruction of the data. It demonstrates that no bug in the code implementation arises.

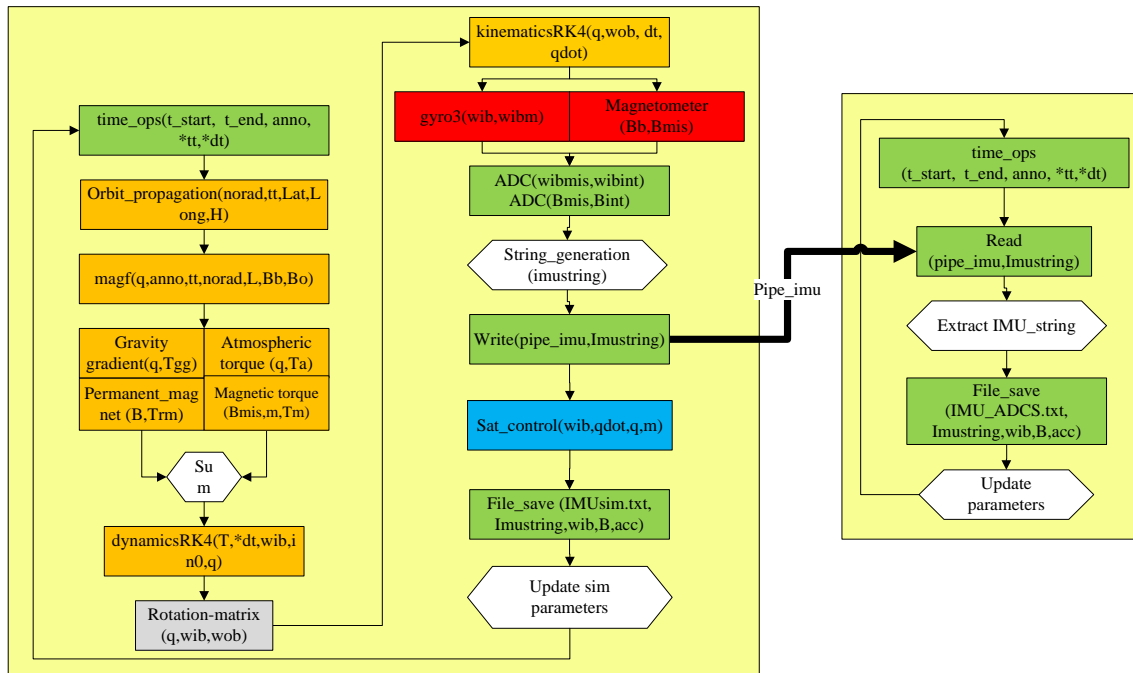


Figure 118: SIL simulation for IMU verification – “models flow” for acquisition and data management

4.4.2.2.2 SIL simulation for ADCS/OBC protocols verification

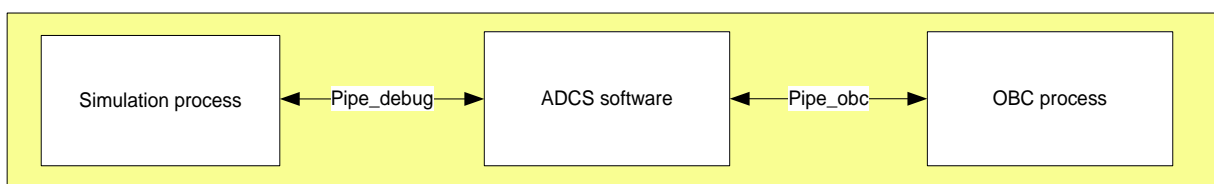
4.4.2.2.2.1 Objectives

The test objectives are:

- the verification of the *ADCS software* capability to extract the strings (called *O2A_string*) received by the OBC process;
- the verifications of the ADCS software capability to format properly the strings (called *A2O_string*) for the OBC.

4.4.2.2.2.2 Setup and configurations

Figure 119 shows the SIL configuration for verification of the communication protocols between ADCS process and OBC process. Three processes are involved: the *simulation process*, the *OBC process* and the process *ADCS software*. The communications between the processes occur through pipes: in particular, *pipe_obc* allows the exchange of data between *ADCS software* and *OBC process* according the relative protocol. *Pipe_debug* is the label of the pipe for the communication of data from *ADCS software* and *simulation process*. *A2O_string* is the ADCS telemetry string formatted by the *ADCS software* process and transmitted to the OBC, *O2A_string* is generated by the OBC-process and sent to *ADCS software* process.



```

setsimarch_ADCS_OBC_SIL.txt x
####
1
tempistica 1
priority 1
path /home/ws1/Dropbox/git/Simulatore/Sim/SIL_ADCS_OBC/OBC/Release/OBC_Process
pipe pipe_obc /home/ws1/Dropbox/git/Simulatore/Ports/Pipes/ rw

####
2
tempistica 1
priority 1
path /home/ws1/Dropbox/git/Simulatore/Sim/SIL_ADCS_OBC/ADCS/Release/SIL_ADCS_OBC
pipe pipe_debug /home/ws1/Dropbox/git/Simulatore/Ports/Pipes/ rw
pipe pipe_obc /home/ws1/Dropbox/git/Simulatore/Ports/Pipes/ rw

####
3
tempistica 1
priority 1
path /home/ws1/Dropbox/git/Simulatore/Sim/SIL_ADCS_OBC/SIM/Release/SIL_ADCS_OBC
pipe pipe_debug /home/ws1/Dropbox/git/Simulatore/Ports/Pipes/ rw

```

Figure 119: SIL simulation for ADCS/OBC protocols verification – processes and interfaces settings

In Figure 120, the “models flow” of the configuration for communication between ADCS and OBC is illustrated. The *simulation process* generates randomly a compatible *IMU_string* and passes it through the *pipe_debug* to *ADCS software* that formats it together other information (quaternion and currents flowing into the MT) according to the protocol rules. The formatted string is sent to the OBC process through the *pipe_obc* and, only for debug, to the *simulation process* (through the *pipe_debug*) that displays and saves the string within the file *A2Osim.txt*. The OBC process receives the string and extracts the information, saving the values in a file *OBC.txt* together with the *string O2A* that has been previously generated and passed to the ADCS process through the *pipe_obc*. The *ADCS software* process receives the strings, extracts and displays the data. The live or post-processing activity consists of analyzing and comparing the *A2O_string* and *O2A string* generated by the “source” and received and handled by the “sink”:

4.4.2.2.3 Results

Different simulation sessions have been carried out in order to verify if the developed piece of software runs properly on ADCS and is able to format /transmit and receive/extract data packets to/from the OBC. The exchange of information between the processes occurs about every 30 seconds so long time session have been performed: 1 hour, 4 hours, 12 hours and 24 hours. The following results are obtained:

1. The ADCS software has reconstructed the correct values for every packet received from OBC process. It has been confirmed, analyzing the values into the strings produced by OBC and the values reconstructed through the specific algorithms by the ADCS. No errors are detected after the comparison.
2. The ADCS software formats all the “A2O” strings according the chosen custom protocol. In fact, loading the data saved in the *A2O_OBC.txt*, it is possible to see that OBC process receives all strings with the right size and the correct header and closer tags. Moreover, the casting of each variable is correctly made by the ADCS and *OBC process* has reconstructed without errors the same values set in the *ADCS software*.

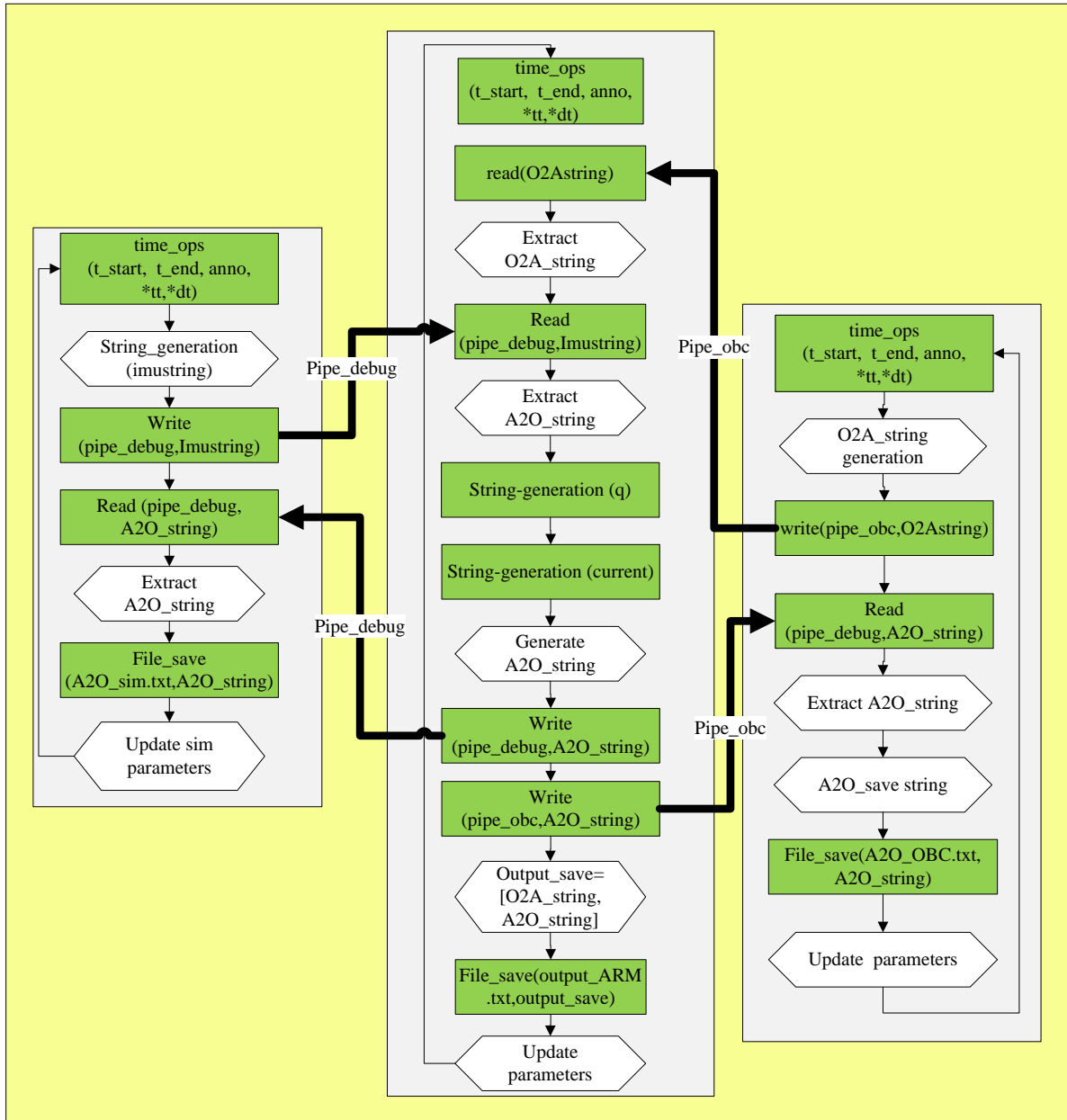


Figure 120: SIL simulation for ADCS/OBC protocols verification - “models flow”

4.4.2.2.3 Complete SIL simulations

4.4.2.2.3.1 Objectives

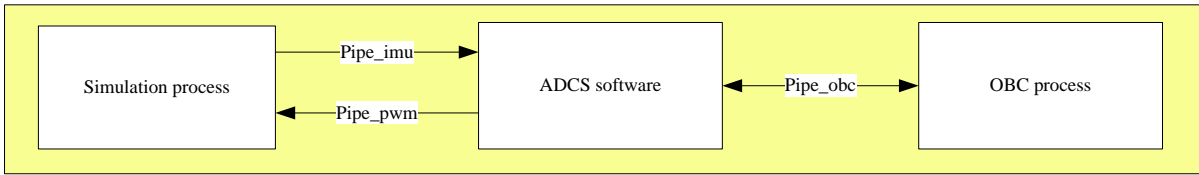
The objective of the test is to verify contemporary the capabilities of the ADCS software:

- To manage all together the IMU stings and the communication with the OBC;
- To perform the determination algorithms and the control strategies

4.4.2.2.3.2 Setup and configurations

Figure 121 sketches the SIL architecture for the purpose of these simulations sessions. Three processes (all running in the simulation unit) and their interfaces can be highlighted. The *simulation process* passes the simulated IMU strings through the *pipe_imu* to *ADCS-process*; The *ADCS-process* communicates *A2O_string* to *OBC process* thanks to *pipe_obc*. On the same interface, OBC passes

O2A_string to *ADCS_process*. Finally, *ADCS process* sends the values of the commands for the actuators to *simulation process* that will use them to calculate the applied control torque.



```

setsimarch_COMPLETE_SIL.txt x
####
1
tempistica 1
priority 1
path /home/ws1/Dropbox/git/Simulatore/Sim/SIL_COMPLETE/OBC/Release/SIL_Obc_Process
pipe pipe_obc /home/ws1/Dropbox/git/Simulatore/Ports/Pipes/ rw

####
2
tempistica 1
priority 1
path /home/ws1/Dropbox/git/Simulatore/Sim/SIL_COMPLETE/ADCS/Release/SIL_ADCS_Process
pipe pipe_debug /home/ws1/Dropbox/git/Simulatore/Ports/Pipes/ w
pipe pipe_imu /home/ws1/Dropbox/git/Simulatore/Ports/Pipes/ r
pipe pipe_obc /home/ws1/Dropbox/git/Simulatore/Ports/Pipes/ rw

####
3
tempistica 1
priority 1
path /home/ws1/Dropbox/git/Simulatore/Sim/SIL_COMPLETE/SIM/Release/SIL_COMPLETE
pipe pipe_imu /home/ws1/Dropbox/git/Simulatore/Ports/Pipes/ w
pipe pipe_debug /home/ws1/Dropbox/git/Simulatore/Ports/Pipes/ r
  
```

Figure 121: Complete SIL simulation - processes settings

From the settings, it can be noted that three different C/C++ codes and relative executable files, after a “simple” compiling, (*est@rII_test*, *est@rII_ARM*, and *est@rII_OBC*) are generated to emulate the actual execution of the software on the ADCS.

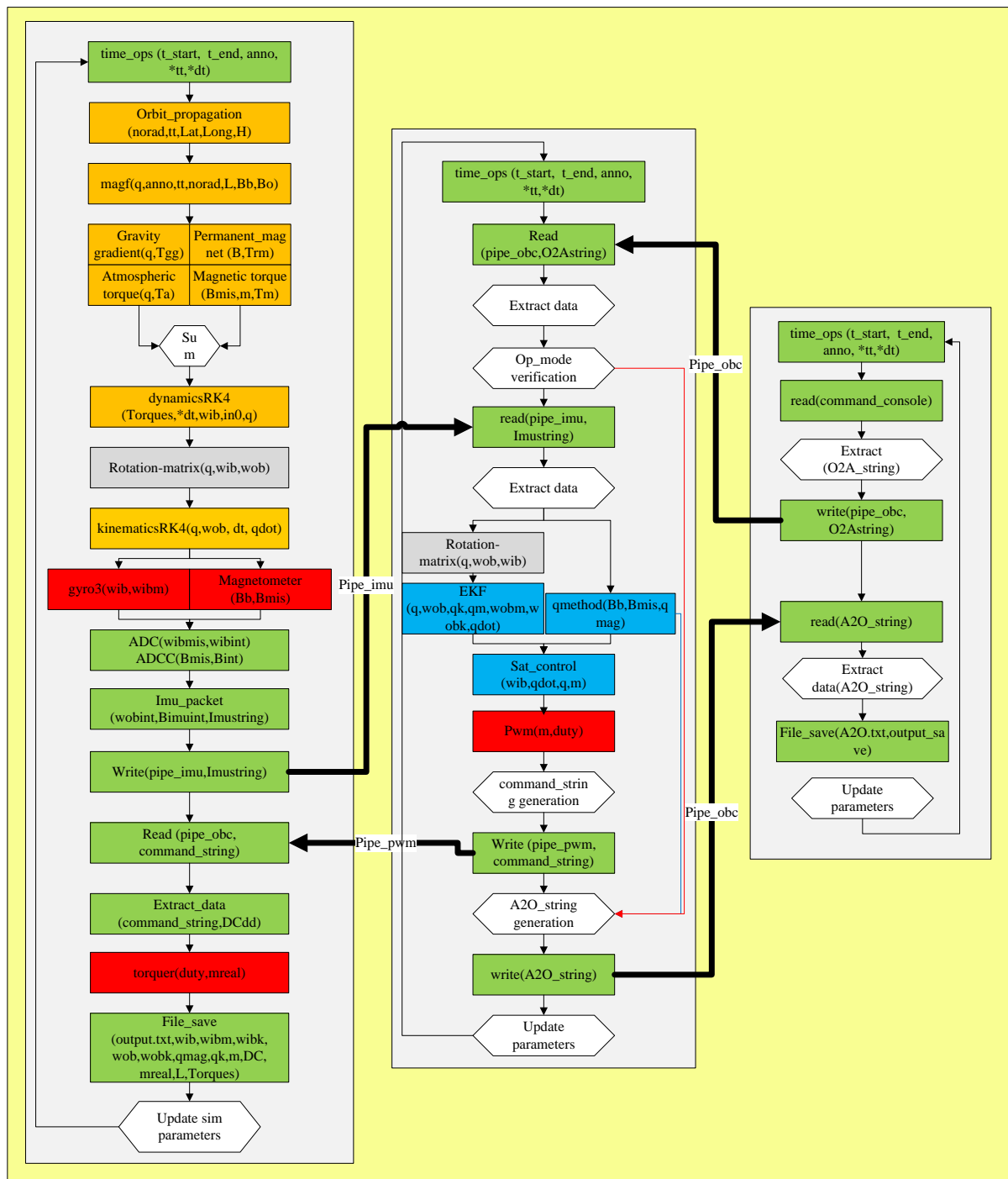


Figure 122: Complete SIL simulations – models flow

Figure 122 shows the “models flows” in the final configuration of SIL simulation. Three flows are present, corresponding to three processes. The *simulation process* (on the left) simulates the environmental conditions and the S/C motion, formats the IMU outputs and passes them to the *ADCS process* through a pipe (called *pipe_imu*), emulating the sensor processor. The *ADCS process* (in the middle) contains the models constituting the software that will run in the ADCS micro-processor. The software manages the modes of operations, acquires IMU’s data, computes the algorithms (already tested in the AIL simulations) for attitude determination and control until the definition of the duty cycle commands for the MTs is done. For the simulation purposes these values are passed to the *simulation process* through the *pipe_pwm*. Moreover a third process (on the right) is setup and refers

to the OBC operations: actually, only the functions related to the communications OBC-ADCS are implemented. In particular, in this process, telemetries and commands exchanged between the two subsystems are taken into consideration. From the control console it is possible to send commands to this process in order to simulate the possibility to change the ADCS mode or to update ADCS parameters from the GCS. This kind of communication is made through a pipe called *pipe_abc*. The following simulation parameters have been chosen and initialized: simulation time= 2 days, step_time= 0,5 seconds, and real time simulation is activated.

4.4.2.2.3.3 Results

The SIL simulations highlighted some problems when the entire software has been assembled. Various bugs have been fixed in the ADCS software: they refer mainly to the insertions of specific functions to handle the control modes (that will be analyzed in the HIL paragraphs), to update some parameters and to synchronize the execution of all the three processes. The modes of operations are quite specific for every mission and it results difficult to generalize that within a function. Moreover, one of the main criticalities (from SIL to HIL) was to synchronize the processes' execution: it means that the communications among the processes (here) and subsystems and simulator processors (for CIL and HIL) shall occur not only in a correct logical way but at precise instant. In order to solve the synchronization problem, delays are introduced properly within the simulation loop.

When all these criticalities are overtaken, new complete simulations ran carrying out to good results:

1. First of all, the interfaces among the processes work properly: in fact, neither ADCS nor OBC processes generate errors on the transmission/reception of A2O and O2A strings. Minor errors occur in the extraction of the respective values but they do not cause great misbehaviors that compromise the results.
2. Both the determination and the control laws works properly and not significant differences can be realized from the AIL simulations. It means that the integration of interfaces and algorithms do not affect the execution of the tasks.
3. IMU string has a rate of good acquisition very close to 100% as well as the command values are always received and used by the simulator.

4.4.2.3 On board microcontroller: choice and tests

The ADCS microcontroller is the RD129 (See APPENDIX E), produced by ELPA s.a.s. [36]. It is based on ARM9 architecture and makes use of a Samsung microprocessor. Among the various peripherals, the used ARM9 I/Os are three serial channels (one only for debug), three timers, and other general-purpose pins. The USART0 allows communication with the IMU, acquiring data (with a specific data protocol) related to the angular velocities, the linear accelerations and the EMF. USART1 allows the communication with the OBC and USART2 is the debug/test line. Each port has a own setting:

- USART0 = 115200 bit/s, 8N1 for the protocol checks;
- USART1 = 9600 bit/s, 8N1 for the protocol checks;
- USART2 = 115200 bit/s, 8N1 for the protocol checks.

The PWM logic signals are generated by the three "timer" outputs of the RD129 properly set by the software. In fact, the ports management is made with virtual files in which the port features are set, defined the PWM frequency, and finally the duty cycle value.

The Operating System for ADCS is Linux Embedded. The kernel (version 2.3.32.xx) has been customized in order to enable only the required I/O ports and related functions, optimizing the performance of the RD129.

The software is written in C/C++ thanks to the simulator that auto-generates the code from the user selection of the models. In the StarSim v1.0, the cross-compiling is still not automatic and the operator

shall perform this activity manually. If the cross-compiling is successfully completed the generated file are manually loaded on the RD129.

At the same time to perform verifications through the methodology, other two codes are compiled and then will run on the simulation unit: the simulation process and the OBC process.

4.4.2.3.1 CIL simulations for PWM settings verification

4.4.2.3.1.1 Objectives

The test aims at verifying the ADCS capabilities to command the timer outputs in order to set the desired PWM duty cycles to drive MT.

4.4.2.3.1.2 Setup & configurations

The configuration (Figure 123) foresees the preparation of two codes: one for the SU (yellow block in the figure) running in the StarSim WS and one for the RD129 micro-processors (blue block in the figure). This software is defined thanks to the simulator and then cross-compiled and loaded on the micro-processor. Moreover, GSE (in violet) are used to power the micro-controller (a power supply unit), to connect the test object (RD129) and the simulator unit, and an oscilloscope to visualize the generated output.

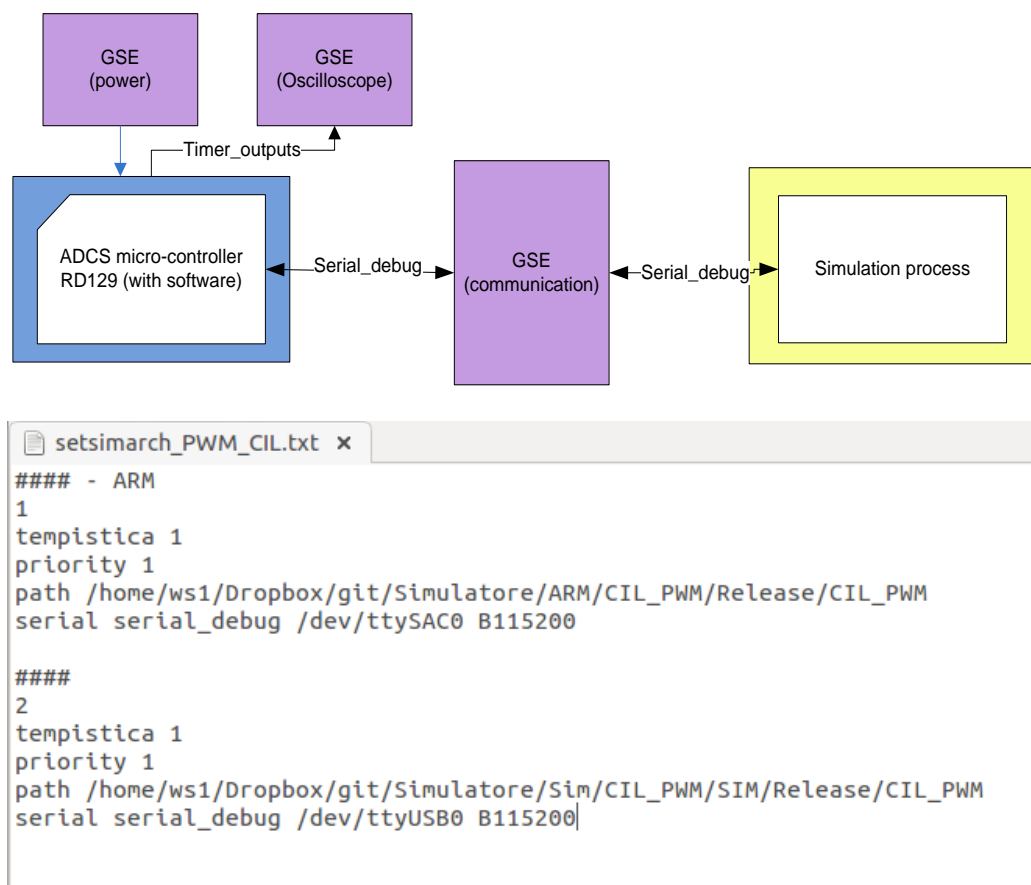


Figure 123: CIL simulations for PWM verification – processes settings

Simulation process generates the PWM commands in order to emulate the attitude control values as computed by the controller. This choice has been made because it allows an easier updating of the values from the user through the GUI. In fact, a second possibility (see next paragraph) is the complete

simulation of dynamics & kinematics and controller but it results overmuch in relation to the stage of this kind of test.

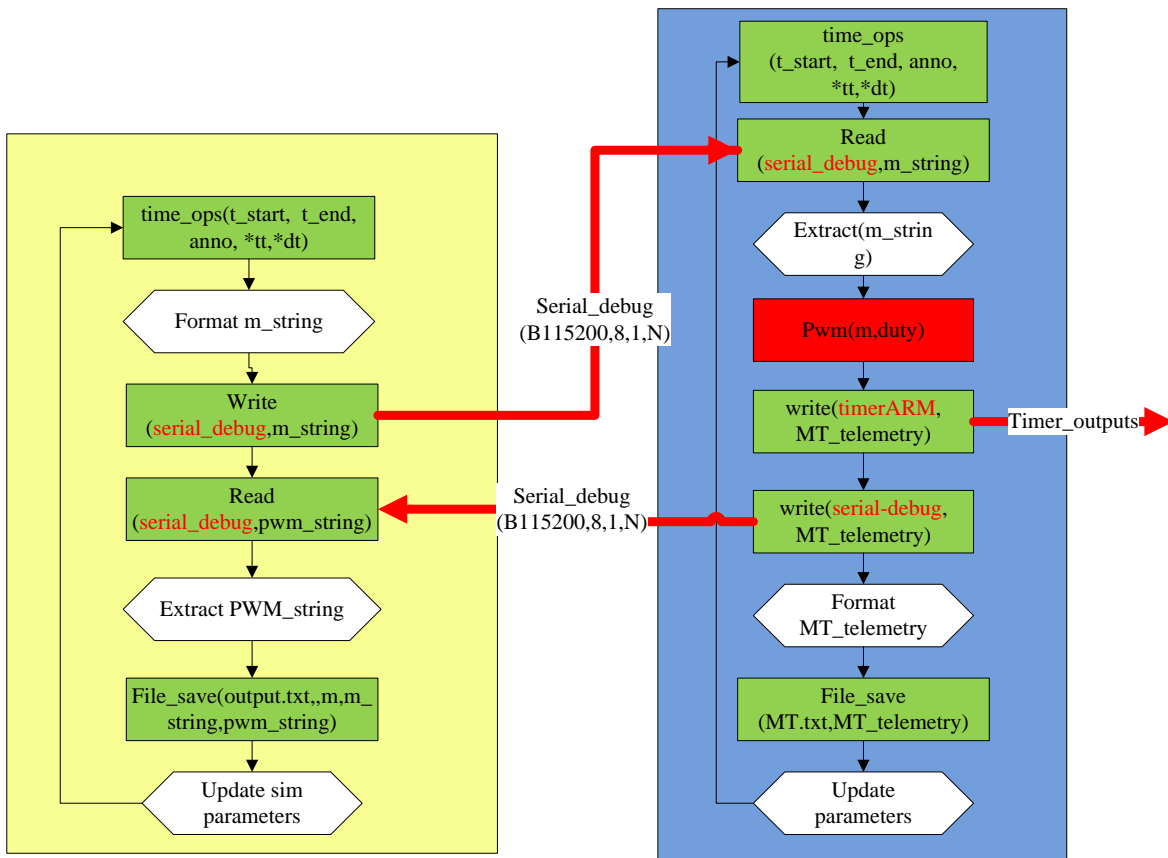


Figure 124: CIL simulations for PWM verification - models flow

The *ADCS software* receives the data through the *serial_debug* and the duty cycle is computed from the commanded values. The *ADCS software* sets these values on *timer_outputs*. The user can verify the right behavior by the use of a oscilloscope that autonomously saves the measurements on files. The set values are both saved in the *PWM.txt* file for future comparisons and sent to simulation process for displaying them (Figure 124). It is expected that the outputs change from the minimum value (and vice versa) varying the time in which the output is at high level voltage wrt a defined time interval (inverse of the frequency of PWM and fixed to 2000 time/s for RD129)

4.4.2.3.1.3 Results

The *ADCS software* sets properly the timer ports output of the RD129: in fact, when the lowest value is set the duty cycle has the minimum values 0.1%, when the highest value is set the duty cycle has the maximum value of 99.9%, and when the center value is set the duty cycle has the value of 50% (it means that the signal remains at the high level for half of the time interval and at low level for the same time). Figure 125, Figure 126, and Figure 127 show the three cases illustrated as plotted by the oscilloscope. Note that the high level voltage is 5 Volt as well as the low level is 0 V.

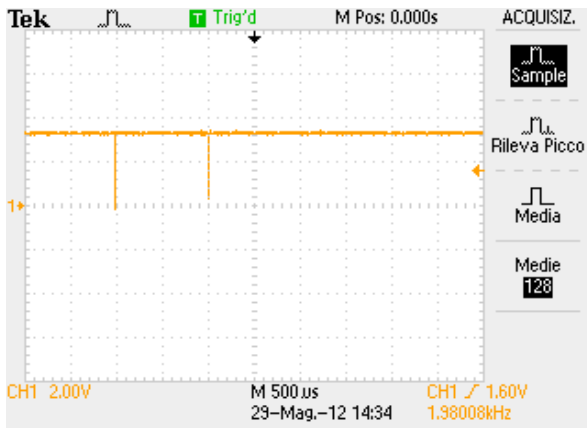


Figure 125: CIL simulations for PWM verification - duty cycle 99.9% (oscilloscope display)

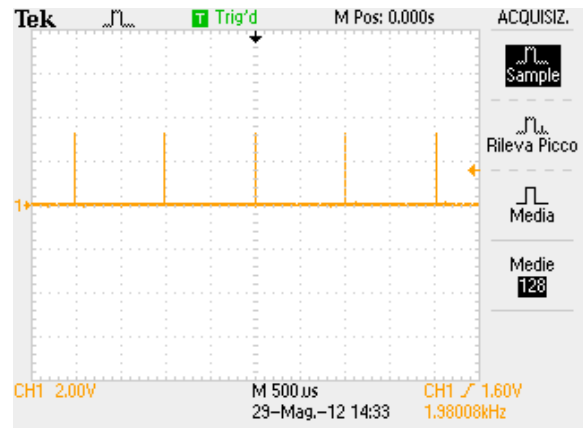


Figure 126: CIL simulations for PWM verification - duty cycle 0.1% (oscilloscope display)

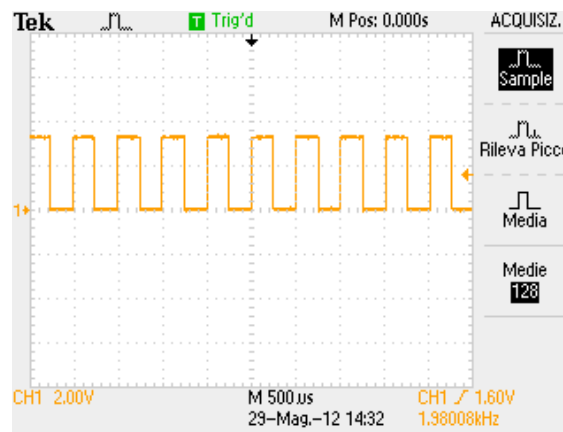


Figure 127: CIL simulations for PWM verification -duty cycle 50% (oscilloscope display)

4.4.2.3.2 CIL simulation for IMU data management

4.4.2.3.2.1 Objectives

The test aims at verifying the capability of the RD129 to acquire IMU's data through the USART0 (*serial_imu*) port and the quality of the link in terms of lost and wrong packets.

4.4.2.3.2.2 Setup and configurations

CIL configuration is used throughout the Design and Development phase to verify the data acquisition and IMU management of ADCS software loaded into the microcontroller ARM9. The configuration (Figure 128) foresees that a single process (called *simulation process*, the yellow block) runs on the Simulation Unit in order to emulate the IMU's output, formats the data according to the datasheet and sends the obtained string through a serial port (*serial_imu*), emulating the real communication between the sensor and the micro-controller. This is possible thanks to the communication GSE (in violet) that adapt the logical levels (from 12V to 5V through a MAX232 chip-based circuit) of the ports. The microcontroller (blue block in the figure) is powered on thanks to power GSE and acquires the string from the serial port (USART0), extracts the data and re-formats it in order to sends it again (through the *serial_debug*) to the simulation process that displays and saves it into the file *output_sim.txt* for future handling and comparison (Figure 129).

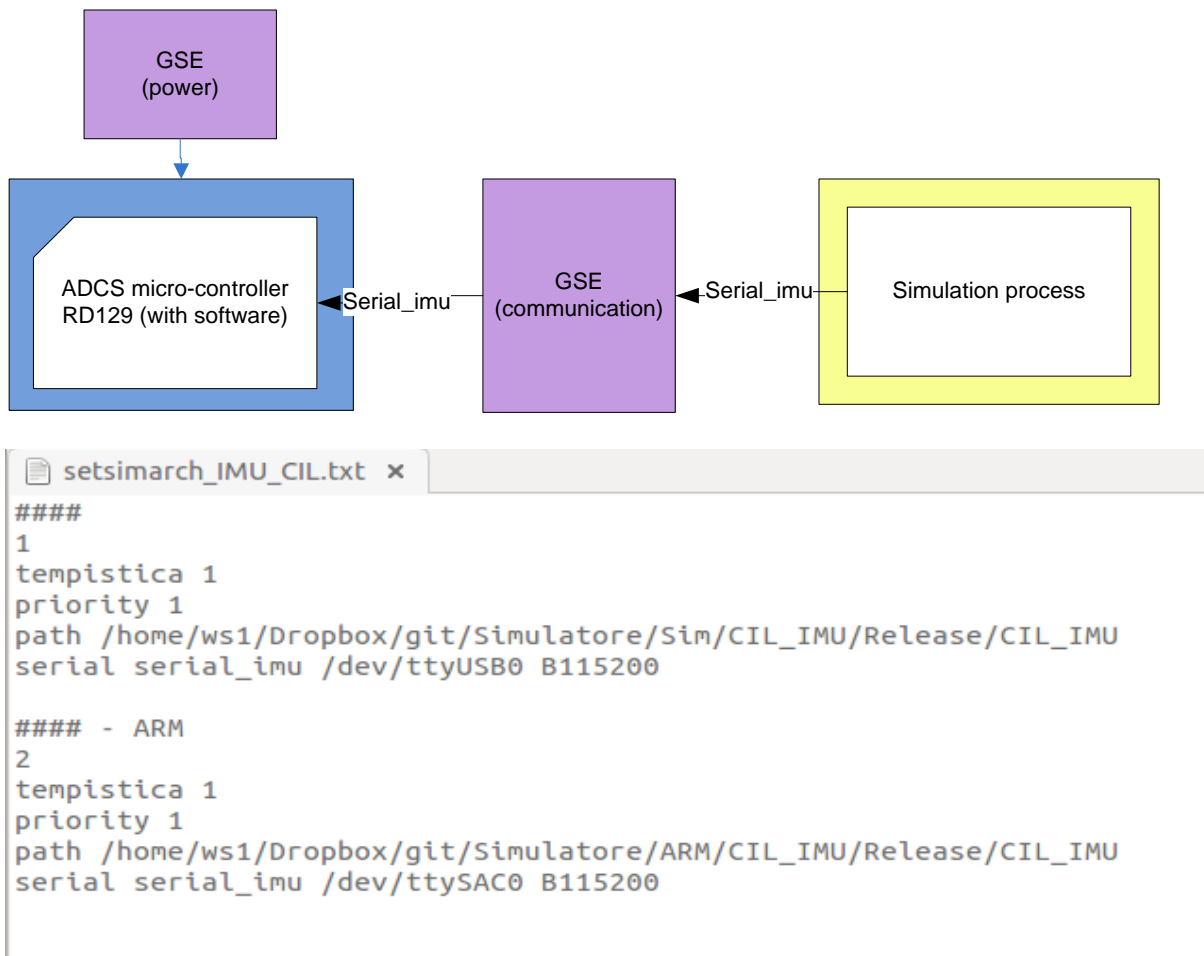


Figure 128: CIL simulations for IMU data management - processes settings

The acquisition capability can be observed by the user during the execution because the RD129 prints the information and sends them through the *serial_debug* connected with the *simulation process*. Moreover, the raw and the extracted values are saved in the file *IMU_arm.txt* on RD129's non-volatile memory. Recovering this file, it is possible to compare the raw and the "real" values with those generated by the simulation process.

Figure 129 shows the "models flow" of the two processes. The *simulation process* (on the left) simulates the system motion and the on orbit environment (orange blocks), the gyroscope and magnetometers behavior (red blocks), the data and signal handling, data saving and time management (green blocks). In particular, there are: the model of the ADC necessary to effectively replicate the sensor behavior, the special function built to simulate the data formatting and packing made by the processor of the sensor. The LINUX generic function *write* allows sending via serial port the *IMU_string* to the *ADCS process*, which is passed to the *ADCS software* process. This process acquires string thanks to the generic LINUX function *read*, validates each string and extracts values; moreover it saves on file (*imu_adcs.txt*) and visualizes information. The post-processing activities consist of reading the two binary files *IMU_sim.txt* and *IMU_ADCS.txt*, saved independently by the two processes and comparing both the binary sequences and the real "physical" data.

These simulation parameters have been chosen:

- Simulation time = 10 minutes/1 hour/4 hours
- Step time= 0.5 seconds
- RT is activated

4.4.2.3.2.1 Results

The results highlight the capability of the microcontroller to acquire data from the serial port. The *simulation process* sends the IMU packets with a frequency of 100 Hz (as the real sensor) and the microcontroller does not lose information after half an hours. Pay attention that the *ADCS process* works at 2 Hz so the main difficult was the synchronization. At this step no problem arises and the strings extraction from the buffer is always completed with success.

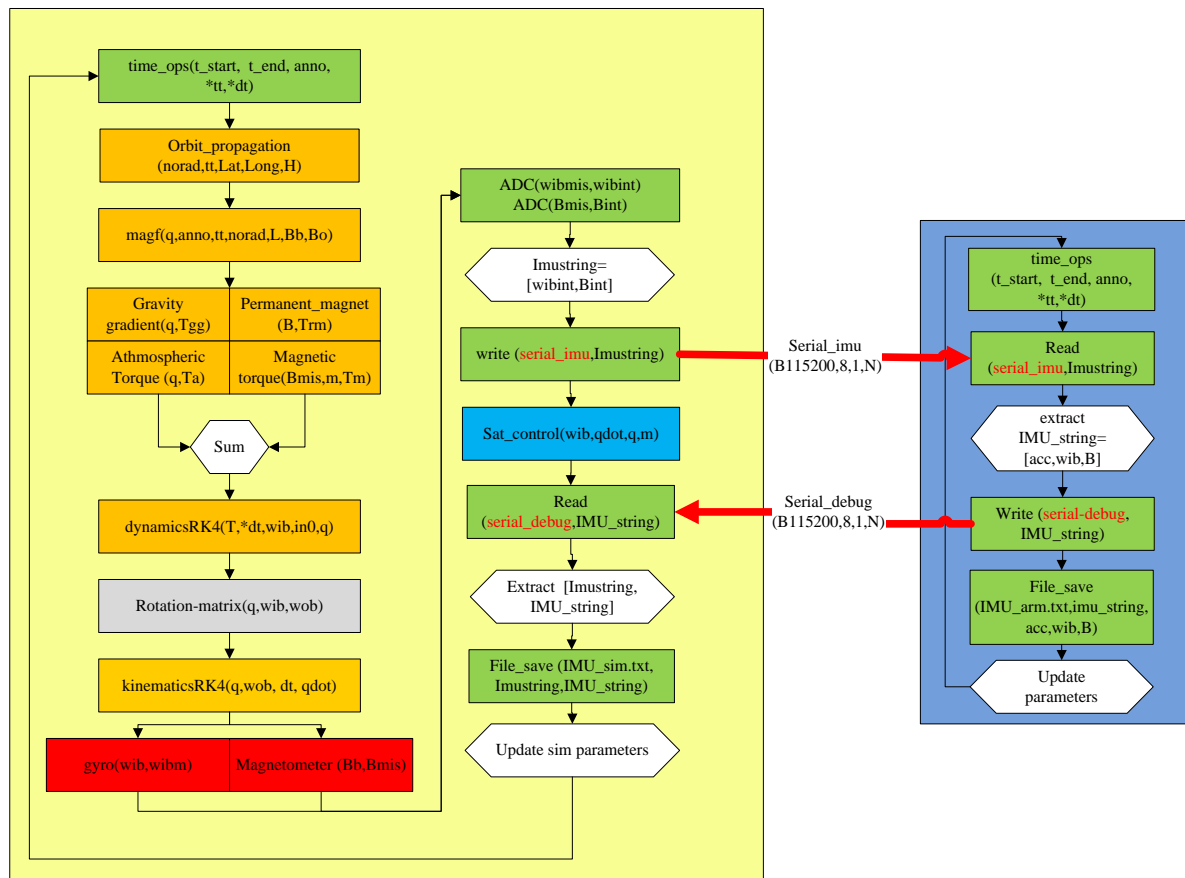


Figure 129: CIL simulations for IMU data management - models flow

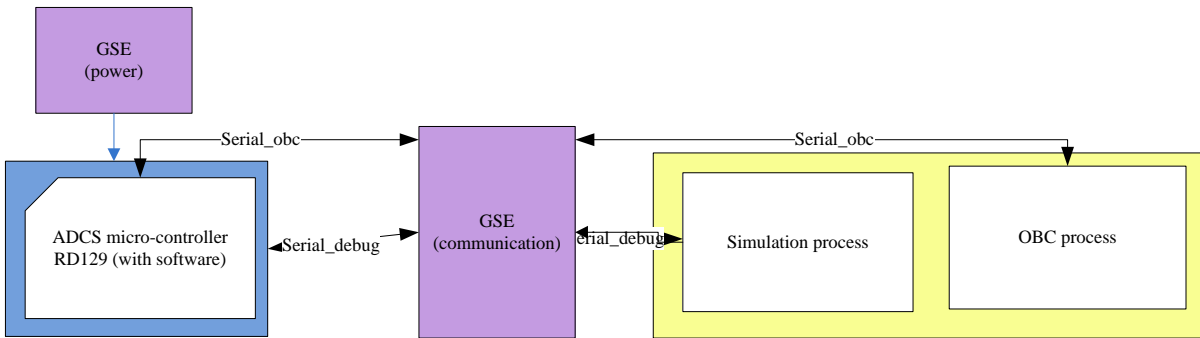
4.4.2.3.3 CIL-Integration – ADCS/OBC communication

4.4.2.3.3.1 Objectives

The test aims to verify the ADCS' data transmission and reception with OBC through the USART0 port of the RD129 and the USART0 port of the OBC micro-processor.

4.4.2.3.3.2 Setup and configurations

Figure 130 shows the CIL configuration adopted to verify the serial communication between ARM9 and OBC processors. Two processes are involved: on the Simulation Unit the *simulation process* and the OBC process. The third element is the RD129 micro-controller (in blu), on which the developed software runs. The RD129 is powered by GSE that provides 3.3V and 5V regulated. Front-end GSE are cables serial/serial 9pins, signal levels adapter based on MAX232 chip. The RD129 microcontroller is connected to OBC process addressed as *serial_OBC* on the codes and physically connected to USART1. The RD129 exchanges information with *simulation process* through the USART2, addressed as *serial_debug* on the code.



```
setsimarch_ADCS_OBC_CIL.txt x
####
1
tempistica 1
priority 1
path /home/ws1/Dropbox/git/Simulatore/Sim/CIL_ADCS_OBC/OBC/Release/OBC_Process
serial serial_abc /dev/ttyUSB1 B115200

#### - ARM|
2
tempistica 1
priority 1
path /home/ws1/Dropbox/git/Simulatore/ARM/CIL_ADCS_OBC/Release/CIL_ADCS_OBC
serial serial_abc /dev/ttySAC0 B115200
serial serial_debug /dev/ttySAC1 B115200

####
3
tempistica 1
priority 1
path /home/ws1/Dropbox/git/Simulatore/Sim/CIL_ADCS_OBC/SIM/Release/CIL_ADCS_OBC
serial serial_debug /dev/ttyUSB0 B115200
```

Figure 130: CIL simulation for OBC/ADCS communication verification - processes and settings

The simulation process generates *IMU_string* and passes it through the *serial_debug* to RD129 that formats it together other information (quaternion and currents flowing into the MT) according to the “A2O” protocol rules. The formatted string is passed to OBC process through the *serial_abc* and, for redundancy, to simulation process (always through the *serial_debug*) that displays and saves the string within the file *A2Osim.txt*. The OBC process receives and extracts the string, saving the values in a file *OBC.txt* together with the *O2A_string* that has previously been generated and passed to the ADCS process through the *serial_abc*. ADCS processor receives the string, extracts and displays the data. (See Figure 131). Pay attention that the ADCS software operates according to the mode set by the OBC (wrt the requests of the GS)

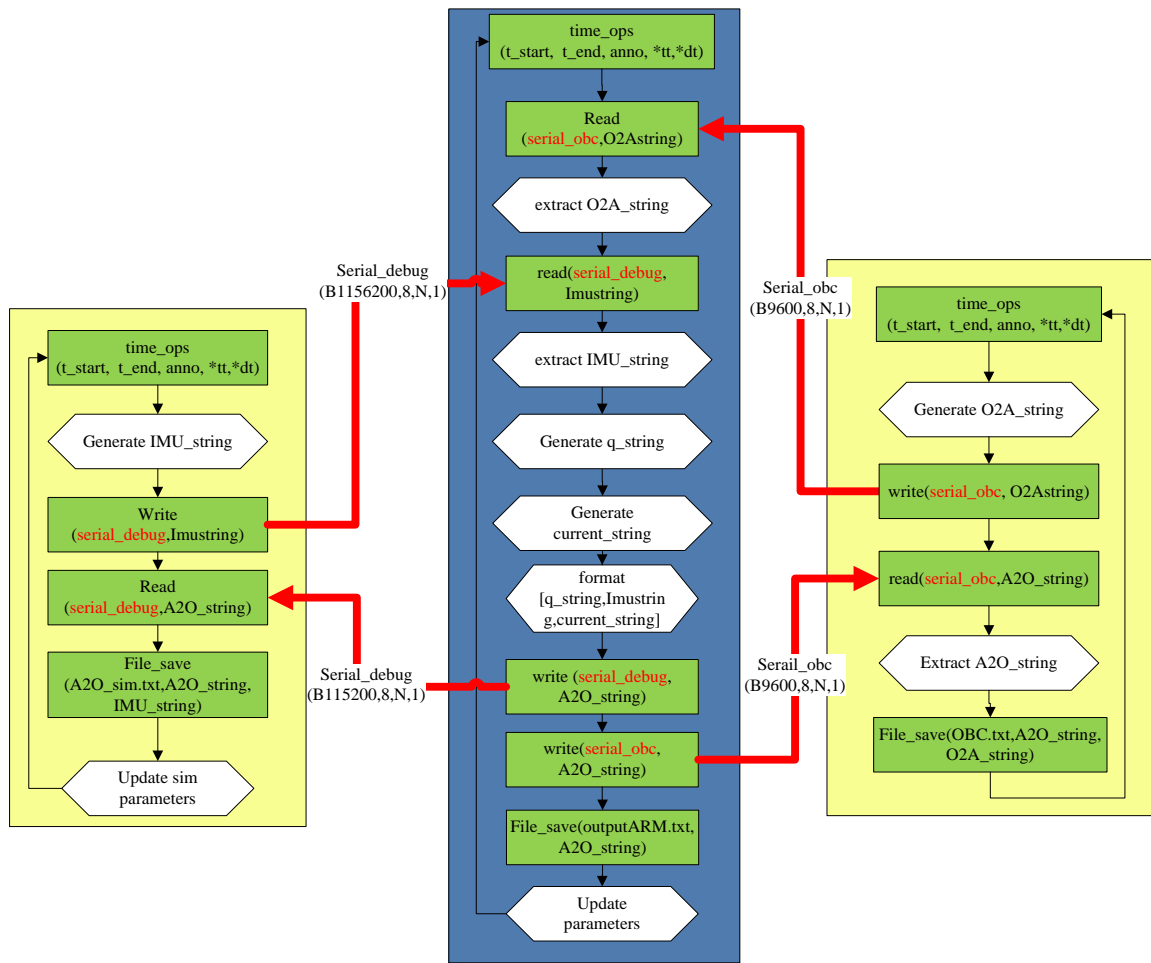


Figure 131: CIL simulation for OBC/ADCS communication verification - models flow

Figure 132 shows the test setup and arrangement using the EM of the ADCS (on the right in the figure) and, also, the development board of the OBC (on the left in the second figure).

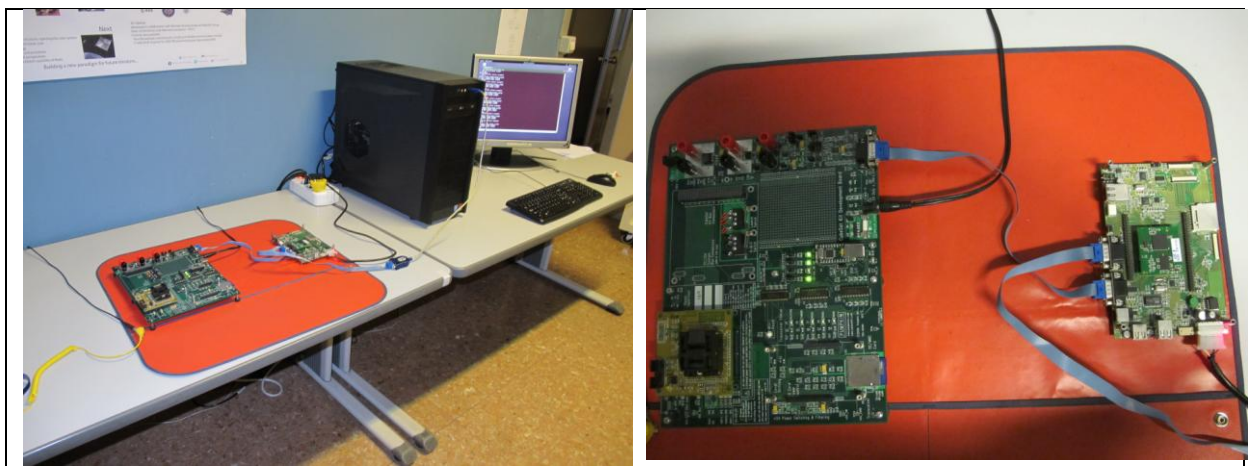


Figure 132: CIL simulation for OBC/ADCS communication verification – test arrangement using EM

4.4.2.3.3 Results

Figure 133 highlights the flag-read variable trend during a simulation session of 2 hours: this flag variable is saved in `O2A_ARM.txt` and is set to 1 when the OBC packet is received and is set 0 if the

packet is not received. No packets are lost at runtime. Moreover, making a comparison among the *O2A strings* within *O2A_ARM* and *O2A_OBC* 1 packet (wrt 240) contains corrupted values.

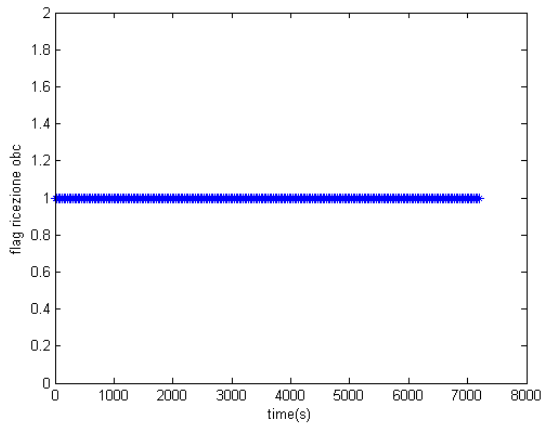


Figure 133: CIL simulations for OBC/ADCS communication verification - *flag read OBC*

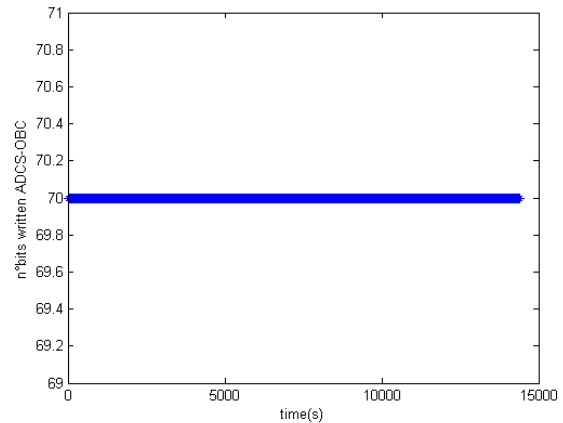


Figure 134: CIL simulations for OBC/ADCS communication verification - number of bits written by the ADCS software on serial-obc port

Figure 134 shows that the ADCS software is able to write the correct number of bits (70) corresponding to the *A2O-string* dimension. OBC has received all these packets but 5 (wrt 480) of them have some wrong values; this consideration derives from the comparison between the *A2O-strings* saved on ADCS and OBC memory, respectively in *A2O_ARM.txt* and *A2O_OBC.txt* files.

4.4.2.4 HW design and test

The ADCS hardware has been designed, developed and built with the fundamental support of Eng. Davide Daprà. The heart of the ADCS hardware is a custom PCB that hosts the RD129 microcontroller, the IMU (Xsens' MT9), and electronic circuits to amplify or acquire the signals. The driving circuits for the MT transform the low level/low voltage [0-5V] command signals (output of the microcontroller) to the higher level voltage applied to the actuators. Other electronic circuits are designed to measure the current (flowing into the MT) data: in particular, these outputs are [0-3.3] V signals and the values are proportional to the current flowing into the MT: in this way it is easy to monitor the power consumption. The microprocessor receives the signals on the GPIO pins and the values are put into virtual files, directly managed by the software. In fact, the ADCS-SW shall be able to initialize and then open, read, write and close the virtual files each times that a new measurement is available. A circuit based on MAX232 chip is used to regulate the signal output of the serial port USART0 because IMU logic and RD129 are different. NO similar circuits are necessary for the serial channel between OBC and ADCS.

The Figure 135 and Figure 136 reports the electronic circuits sketches. First figure highlights the connections of the 120 pin of ARM9, the connections of the 104 pin of the satellite bus and other circuits: the levels adapter circuits for the interface between the RD129 and the IMU is signed as "Magnetometro" and is based on the MAX3232 chip that adapts the 12 V signals of the IMU TX line to 3.3 V accepted in input by the microcontroller. The other presented circuits have the same function of signal adapter but they are not involved directly in A-ADCS subsystem

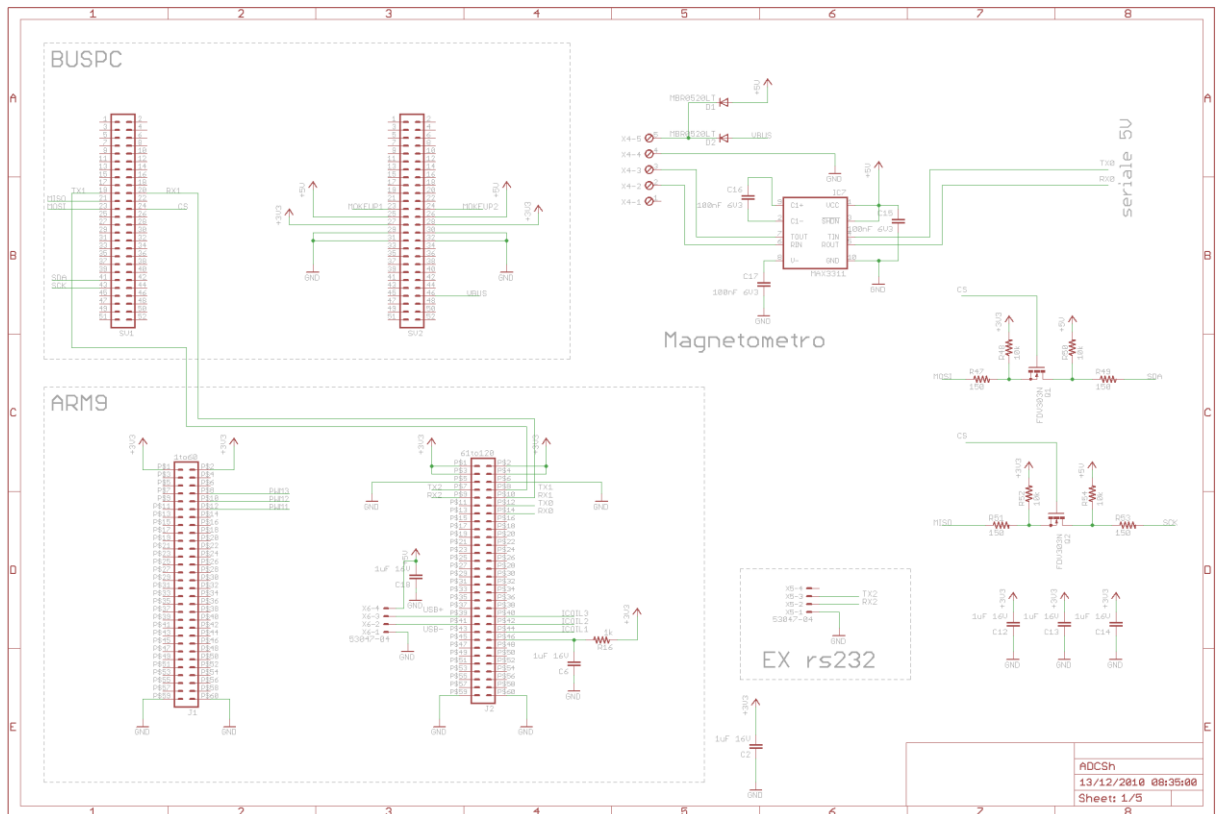


Figure 135: Schematic of the ADCS board (I)

Figure 136 depicts two type of circuits: the PWM logic circuit and the current sense circuits. The command signal enters in PWM-1: it is a square signal with varying duty cycle. After IC1A the signal passes to a not inverting and not inverting stage (the circuits related to AD8397 amplifiers). When the DC is 50% a mean value voltage of 2.5 V passes through the stage and no potential difference arises on the torque connected to “53047-04” connector. If a low DC is applied, a voltage close to 0V is the output of the not inverting stage and a voltage close to 5V is in output of the inverting stage, generating a potential difference on the MT in which will flow a current from X1-4 to X1-1. Vice versa, if a high DC is commanded, a voltage close to 5V is on the not inverting stage and a voltage close to 0V on the inverting stage, generating a potential difference on the MT into which a current will flow from X1-1 to X1-4. The current sense circuit is based on amplifying differential stage. According to the current flowing into MT a different voltage is applied to R68. The potential difference passes through the LT1498 amplifier whose output is proportional to the input. In particular, if no current flows the output is 1.6 V, if current flows from X1-4 to X1-1 then the output tends to 0V (corresponding to maximum in absolute value), and if current flows from X1-1 to X1-4 then the output tends to 3.3V (corresponding to maximum in absolute value). In this way it is possible not only to evaluate the power consumption but also the direction of the current flowing. All the circuits in Figure 136 are replicated three times, one for each MT command channel.

Finally, a linear technology is adopted instead a switching technology because the currents are very low and the MT are often powered off during the mission. In this condition the linear technology is more efficient in terms of power consumption because no currents flow in the circuit when it is switched off.

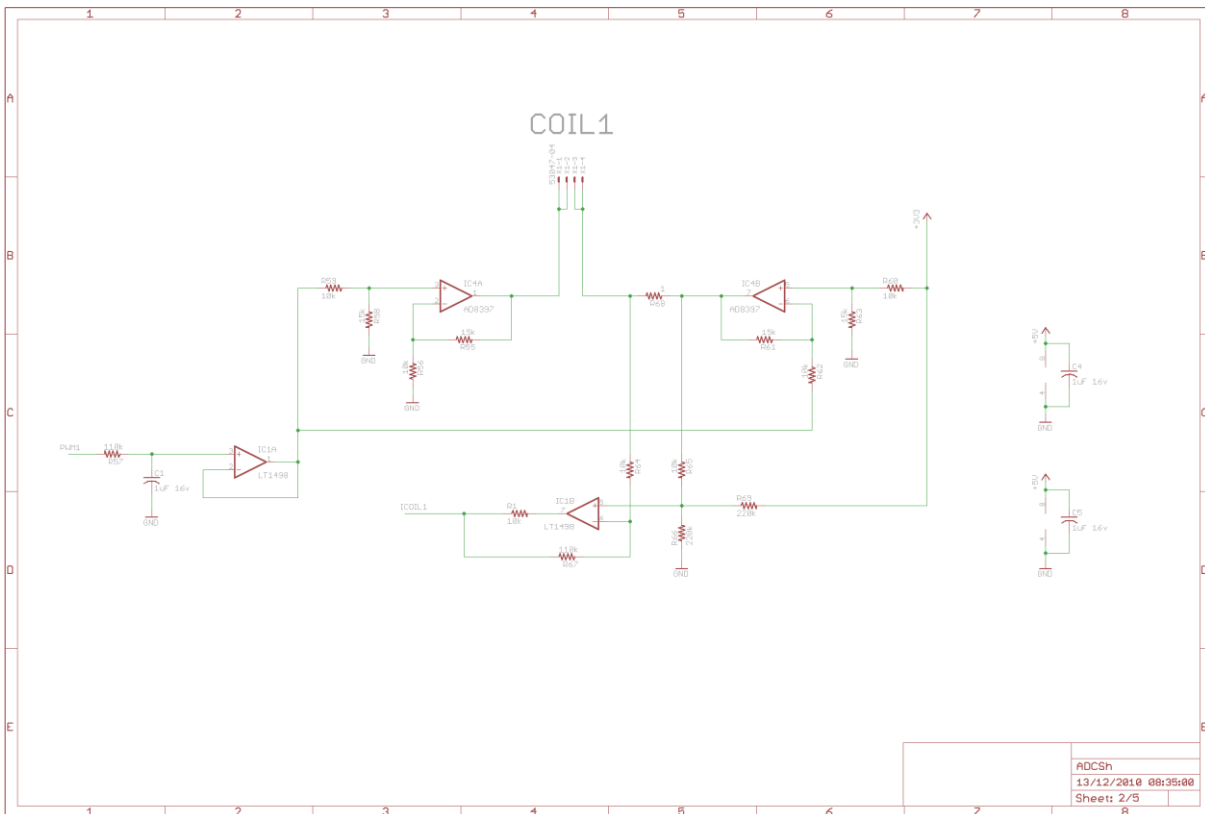


Figure 136: Schematic of the ADCS board (II)

The Figure 137 shows (on the left) the printed board view (made in Eaglecad [35]) in which all the circuits, chip and passive elements take place. The red elements stay on the top of the board, the blue ones stay on the bottom. The same figure presents on the right a picture of the FM model board during a continuity check test: on the lower part of the picture it is possible to see the 104-pin connector (produced by Samtec)[39]; in the right there is the RD-129 and on the left there is the IMU sensor (produced by Xsens)[40].

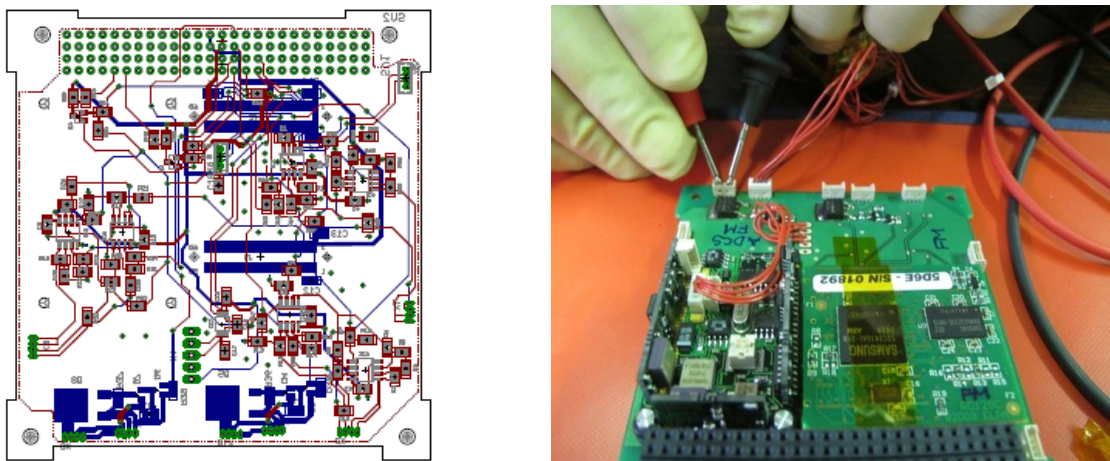


Figure 137: ADCS board, view from Eaglecad and FM during basic electrical tests



Figure 138: Magnetic torquers (MT)

Also the Magnetic Torquers have been made in-house at the StarLab of Politecnico di Torino. Their design and construction is not a topic for the thesis, so only a picture to show (Figure 138) the final product is here included. [42]

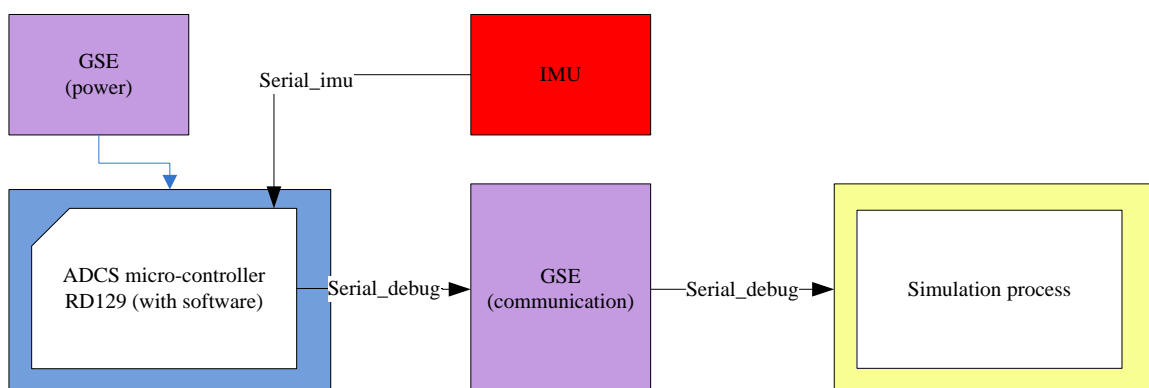
4.4.2.4.1 HIL simulation for IMU functional test

4.4.2.4.1.1 Objectives

The test aims at the verification of ADCS microcontroller capability to acquire and handle the data from real IMU.

4.4.2.4.1.2 Setup and configurations

Figure 139 shows the HIL configuration for real IMU acquisition and calibration. One process runs on the Simulation Unit (yellow) and receives information through the *serial_debug*. The ADCS is equipped with the RD129 microcontroller and the IMU. The micro-controller receives the IMU raw through the *serial_imu* port (USART0) and passes the output of the data handling function to the *simulation* process. The ADCS board must be provided with 3.3V, 5 V and a voltage in the range [7-8.2] V regulated using a power supplier. GSE Frontend as cables RS232 9pins and MAX232 adapter circuits are required to complete the setup.



```

setsimarch_IMU_HIL.txt x
####
1
tempistica 1
priority 1
path /home/ws1/Dropbox/git/Simulatore/Sim/HIL_IMU/Release/HIL_IMU
serial serial_debug /dev/ttyUSB0 B115200

#### - ARM
2
tempistica 1
priority 1
path /home/ws1/Dropbox/git/Simulatore/ARM/HIL_IMU/Release/HIL_IMU
serial serial_imu /dev/ttySAC0 B115200
serial serial_debug /dev/ttySAC1 B115200

```

Figure 139: HIL simulations for IMU functional test: processes settings

The “models flow” (Figure 140) illustrates that the real IMU sensor provides the information under data raw form through the *serial_imu* port. The software loaded in the RD129 reads the data and inserts them in the vector “Imustring” from which it extracts the telemetry related to accelerations, angular velocities, and EMF with respect to the sensor frame (that will be assumed coincident with the body frame). These values are passed to the *simulation process* to display and save them for future comparison.

Changing IMU orientation and the temperature conditions, different measurements are derived and saved: the acquisitions allow estimating the real values and reconstructing the ADC error, the bias drift and the misalignment.

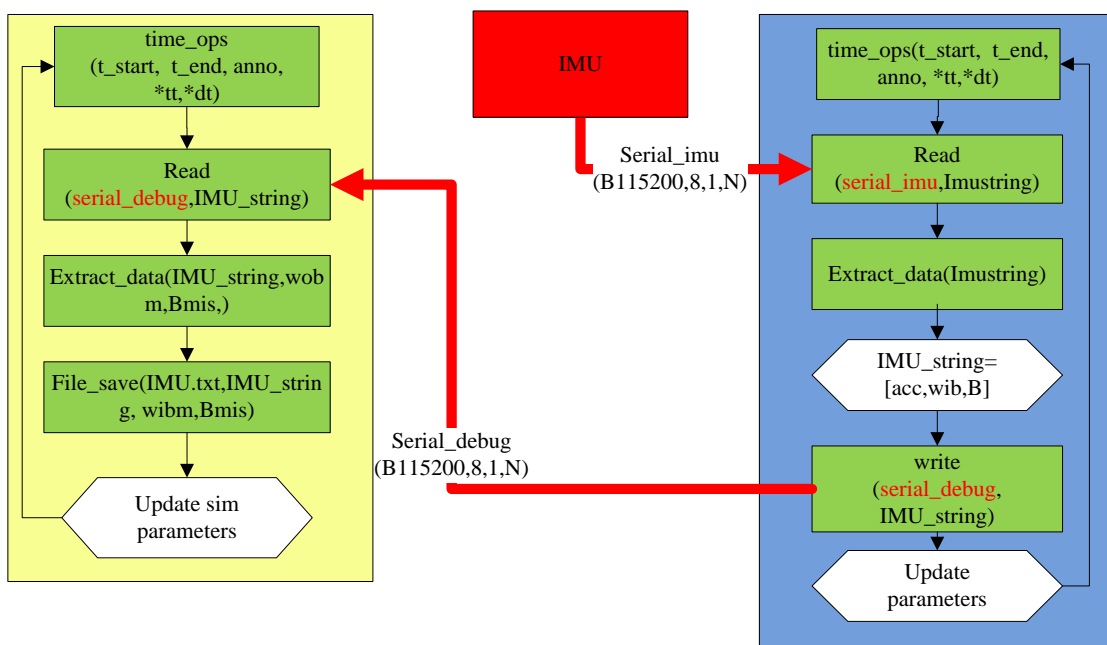


Figure 140: HIL simulations for IMU functional test -models flow

Figure 141 contains two pictures of the arrangement of the test: the first shows an overview with the simulator (Simulation Unit and Control Console) on the right, the power supplier in the middle and the ADCS board (FM) both on the left and in the second picture. The IMU is mounted near the RD129

mounted with the gyroscopes as close as possible to the CoM. A Frontend GSE adapts the serial communication level among the ARM9 and the Simulation Unit.

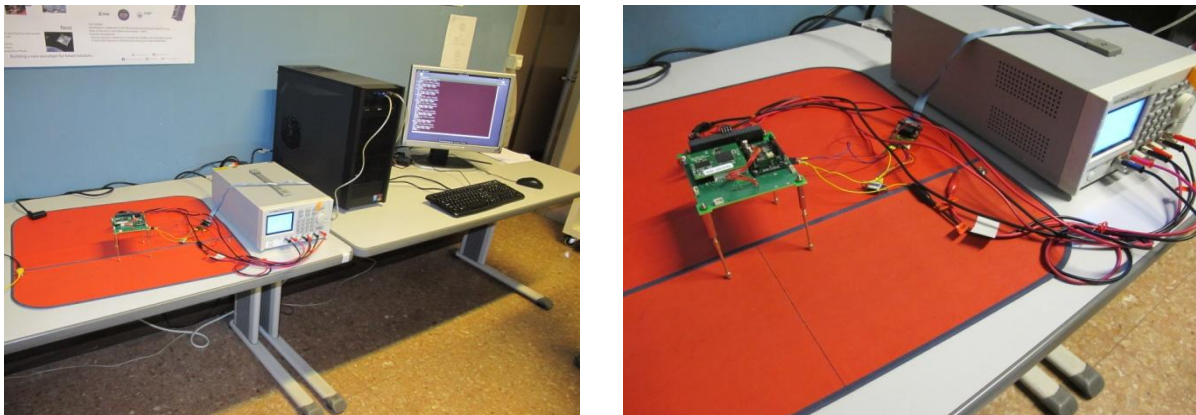


Figure 141: HIL simulations for IMU functional test – arrangement of the test using FM unit

Figure 142 shows the generated code for the *simulation process*.

The first window show the initialization code with the included libraries and database, the log files settings, and the variables initialization. The second window shows the functions and the models involved for time management, data raw reading from the serial port, data extraction and file saving management

```

HIL_IMU.c
Name : Process.c
#include "headers.h"
#include "function_header.h"
#include "port_libs.h"

int main(int argc, char* argv[]) {
    /*
     *
     *      INITIALIZATION
     *
     *
     */

    struct pipe *pipe_ptr;
    struct serial serial_debug, *serial_ptr;
    int n_pipes = 0, n_serials = 1, i = 0;

    if (argc < 2) {
        printf("Wrong number of arguments passed.");
        exit(EXIT_FAILURE);
    }

    i = 0;

    serial_ptr = &serial_debug;
    open_serials(argv[i + n_pipes + 2], serial_ptr);
    i++;

    /*
     *
     *      VAR POOL
     *
     *
     */

    struct timeval end, start;
    double wobm[3] = {0},
        anno = 0, tstart = 0, tt, dt, tf[6],
        Bmis[3] = { 0 }, tt1;
    unsigned char imu_string[25] = { 0 };
    int dd, contatempo;

    system("rm -r -f /home/wsl/Dropbox/git/Simulatore/e-st@rII_test/Debug/LogFiles/");
    system("mkdir /home/wsl/Dropbox/git/Simulatore/e-st@rII_teSt/Debug/LogFiles/");
}

```

```

HIL_IMU.c
while (1) {
    gettimeofday(&start, NULL);
    time_ops(start, end, anno, tstart, &tt, &dt, tf);
    dd = read(serial_debug.serial_fd, imu_string, 150);
    if (dd == -1) {
        printf("Error in reading from port!");
        exit(EXIT_FAILURE);
    }
    printf("%d: bytes read: %d\n", getpid(), dd);
    extract(imu_string, wobm, Bmis);
    file_save(&contatempo, imu_string, wobm, Bmis, tf, &tt1);
}
}

```

Figure 142: HIL simulations for IMU functional test - C++ code skeleton

4.4.2.4.1.3 Results

A high number of simulations has been performed for the IMU acquisition in CIL because it resulted very difficult to guarantee a sufficiently reliable communication between IMU and the ADCS board because the sensor provides 100 complete strings every second while the ADCS process runs at 2 Hz: this fact causes an overflow for the buffer variable devoted to gather all the IMU packets. The second problem occur because often the extracted string from the buffer does not corresponded to the latest and the obtained values were not the real time values. For these reasons, a change in the ADCS software has been necessary: the IMU acquisition is now in charge of a thread (see chapter 3.3 for the definition of thread), a sub-process that runs at the same speed of the IMU, acquires all the strings and passes the last ones to the main process (the *ADCS process*) when it is required. This solution allows giving a good affordability on the IMU acquisition task. In fact, the number of wrong or lost IMU's packets by ADCS process is 3 every 100. Actually, it is not the best solution but it is sufficient to guarantee that the on board algorithms were able to determine attitude from magnetometers and/or gyroscopes measurements.

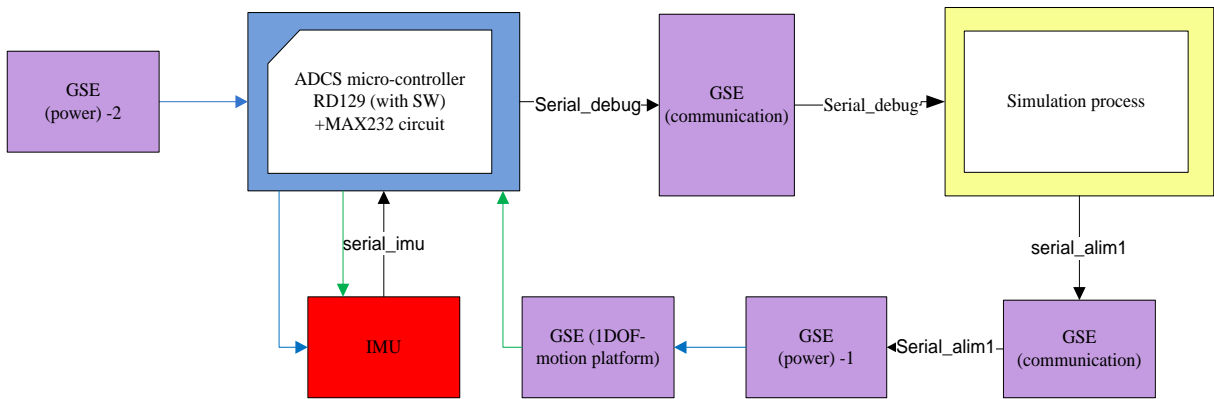
4.4.2.4.2 HIL simulation for IMU calibration

4.4.2.4.2.1 Objectives

The test [43] aims at the calibration of the IMU and the validation of the data acquired by RD129.

4.4.2.4.2.2 Setup and configurations

Figure 143 shows the HIL configuration for real IMU calibration. The elements involved in the test are the simulation process (yellow frame), running on the Simulation Unit, the real IMU sensors, the MT9 produced by Xsens (the red block), the RD129 microprocessor (in the blue frames) and different types of GSE (violet blocks). The *simulation process* commands the voltages and currents of a power supplier (GSE-1 (power) blocks) through the *serial_alim1* port and receives, for redundancy, the data acquired by the ADCS board through the *serial_debug* channel and thanks to Frontend GSE based on MAX232 serial logic levels adapter. The power supplier (GSE-1) is used to control the motor of a 1-DOF platform used to rotate a plate on which the test object (IMU and ADCS board) is arranged with particular careful in order to have a mechanical connections (the green line) as good as possible both between the board and the plate and between the sensors and the board on which the sensor is mounted (thank to four M2 aluminum screws). IMU provides data to ADCS-board through the *serial_imu* port (USART0). The GSE-2 (power) supplies regulated voltages to ADCS board through the main satellite bus and IMU is powered directly from the ADCS-board).



```

setsimarch_IMU_HIL_CALIBRATION.txt x
####
1
tempistica 1
priority 1
path /home/ws1/Dropbox/git/Simulatore/Sim/HIL_IMU_CALIBRATION/Release/HIL_IMU_CALIBRATION
serial serial_debug /dev/ttyUSB0 B115200
serial serial_ps1 /dev/ttyUSB1 B9600

#### - ARM
2
tempistica 1
priority 1
path /home/ws1/Dropbox/git/Simulatore/ARM/HIL_IMU_CALIBRATION/Release/HIL_IMU_CALIBRATION
serial serial_imu /dev/ttySAC0 B115200
serial serial_debug /dev/ttySAC1 B115200

```

Figure 143: HIL simulations for IMU calibrations - processes settings

The “models flow” is illustrated in Figure 144. The real IMU sensor provides the data raw through the *serial_imu* port to USART0 of RD129. The ADCS software reads IMU data (with the generic LINUX function *read*) and extracts the telemetry related to accelerations, angular velocities, and EMF values wrt the sensor frame (that will be assumed coincident with the body frame). These values are saved in *IMU_adcs.txt* file passed to the *simulation process* to display and save them for future comparison. The *simulation process* manages time and synchronizations and contains a function written ad hoc for this test. This function is an algorithm studied to provoke controlled rotation of the plate mounted on the 1-D platform: in particular, different voltages are set and consequently the plate rotate with different angular velocities allowing testing the capabilities of the gyros. Finally, the simulation process receives the IMU data through the *serial_debug* port (USART2) and saves it in the file *IMU.txt*.

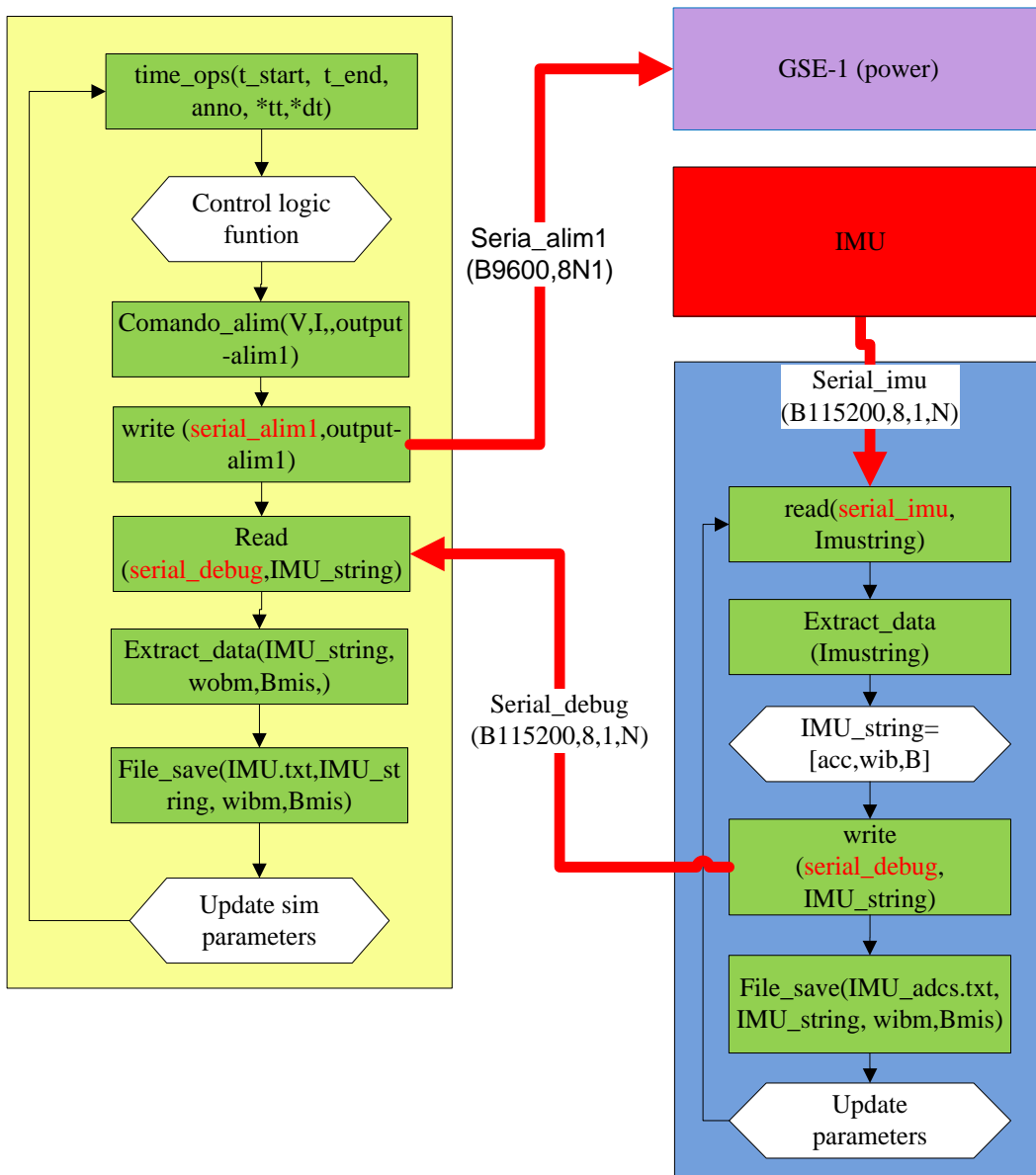


Figure 144: HIL simulations for IMU calibration - models flow

Figure 145 contains three pictures of the arrangement: the first shows an overview and details of the simulator (Simulation Unit and Control Console) on the right, the power suppliers in the middle and the ADCS board (FM) mounted on the 1D platform. The second and third ones are focused on GSE and test object (the ADCS board with IMU).

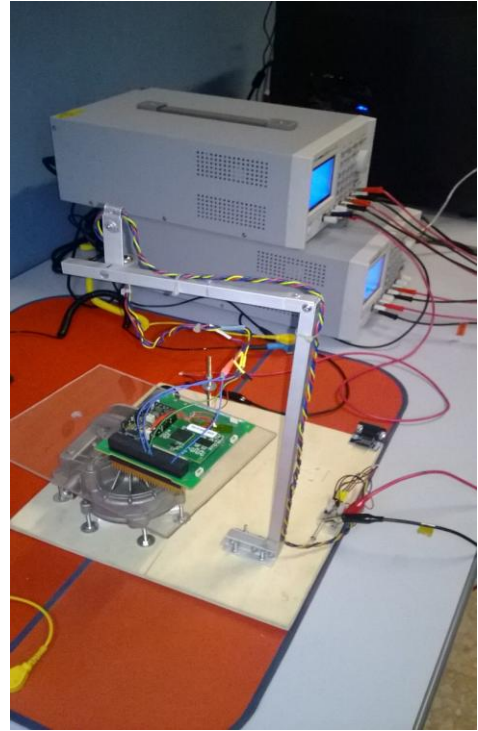
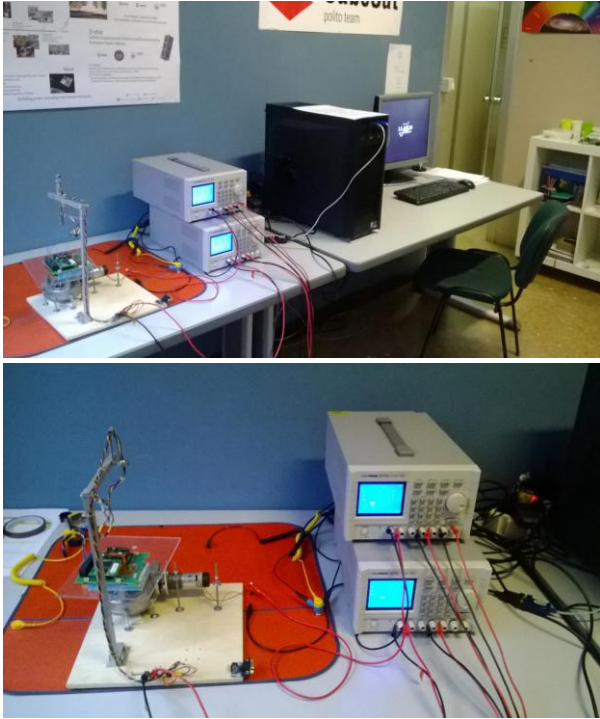


Figure 145: HIL simulation for IMU calibration – arrangement of the test with QM

4.4.2.4.2.3 Results

The results of the test are shown in Figure 146, Figure 147, and Figure 148. Gyroscopes' behaviour is quite similar to that expected: small variations could be related to the homemade rotating plate, and surely with a more accurate GSE, results will reflect exactly the expected ones.

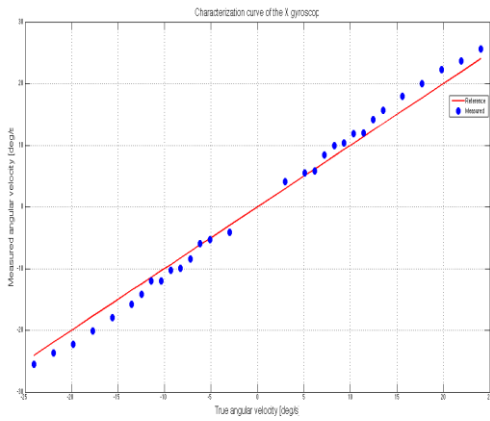


Figure 146: HIL simulation for IMU calibration - X-axis gyroscope characterization

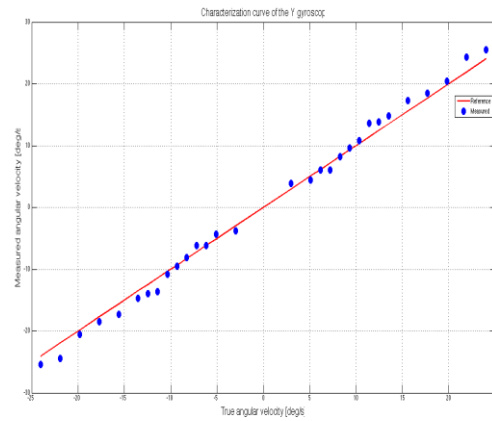


Figure 147: HIL simulation for IMU calibration - Y-axis gyroscope characterization

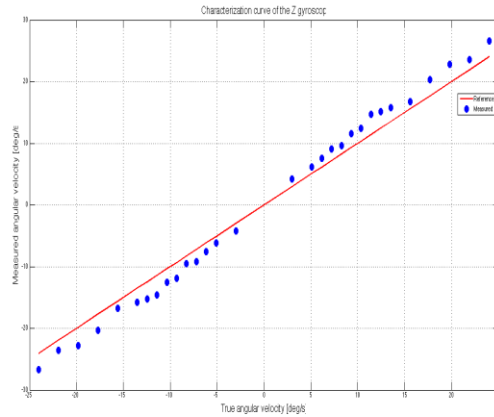


Figure 148: HIL simulation for IMU calibration – Z-axis gyroscope characterization.

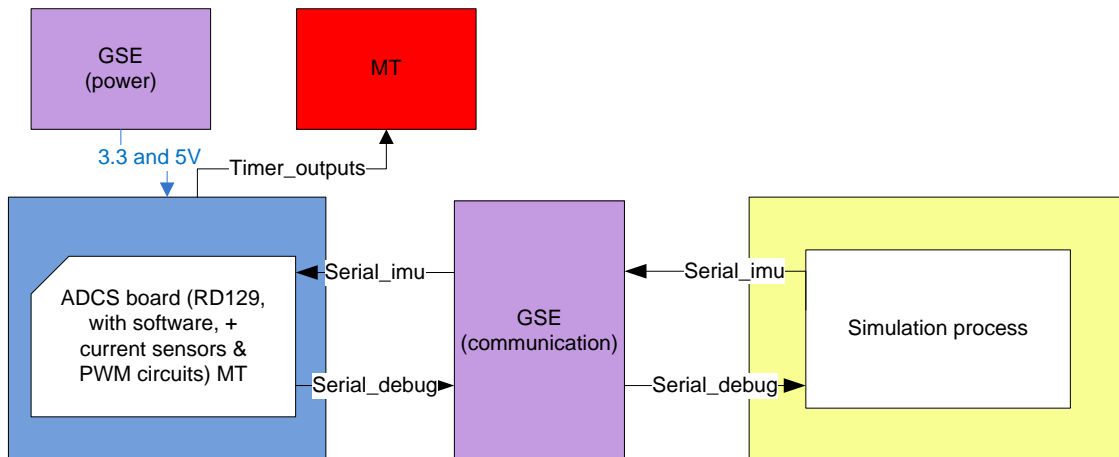
4.4.2.4.3 HIL simulation for PWM and current sensors calibrations

4.4.2.4.3.1 Objectives

The test aims at the verification of the PWM commands generation on the timers RD-129's ports and the measurement of the currents flowing into the actuators.

4.4.2.4.3.2 Setup and configurations

The configuration (Figure 149) foresees that the *simulation process* generates the commands in order to emulate the attitude control values as computed by the controller. The ADCS software receives the data through the *serial_debug* and the duty cycle is computed from the defined values. The ADCS software sets these values on *timer_outputs* connected to the PWM logic circuits in charge of settings the corresponding currents flowing into the MTs.



```
setsimarch_PWM_HIL.txt x
#### - ARM
1
tempistica 1
priority 1
path /home/ws1/Dropbox/git/Simulatore/ARM/HIL_PWM/Release/HIL_PWM
serial serial_debug /dev/ttySAC0 B115200

####
2
tempistica 1
priority 1
path /home/ws1/Dropbox/git/Simulatore/Sim/HIL_PWM/SIM/Release/HIL_PWM
serial serial_debug /dev/ttyUSB0 B115200
```

Figure 149: HIL simulations for PWM and current sensors calibration – processes settings

The user can verify the right settings thanks to an oscilloscope that autonomously saves the measurements on files. Another circuit “transforms” these currents, thus generating output signals within the range [0-3.3] Volts. They are acquired by the microcontroller on GPIO pins through *gpio-adc* channels and, through an internal 10-bits ADC, that gives proportional values that are transmitted to the simulation process together with the set PWM’s values and then saved (file *MT.txt*) for post-processing use. In the same way, the *simulation process* acquires the MT telemetry string, displays, and saves (file *output.txt*) the extracted values.

The Figure 150 illustrates the “models flow” of the simulation process (in the yellow frame) and the ADCS software executed by the RD-129 (blue frame). In the Simulation Unit, the generated code derives from the models in charge of 1) managing simulation time and synchronization, 2) generating PWM string, and 3) using the LINUX generic function *write* the string is sent to the ADCS board via serial communication. SU reads the MT telemetry (deriving from the current sense circuits), extracts them, and, finally, saves all the sensible data of the session (MT telemetry, duty cycles, currents) on the file *output.txt*. The flow in the blue frame starts with UTC management function; then ADCS-software reads the strings with PWM commands, extracts them, computes the duty cycle (with the relative algorithm), and sets the values in the *co* respective virtual files. In the loop, there are the real hardware in terms of PWM circuits, MTs, and current sense circuits. The current values are saved in the virtual file that the ADCS-software reads and then puts into the file *MT.txt*. Moreover, the values are sent to the simulation process for debug and successive comparisons and other post-processing activities.

The generated code for the *simulation process* is reported in Figure 151.

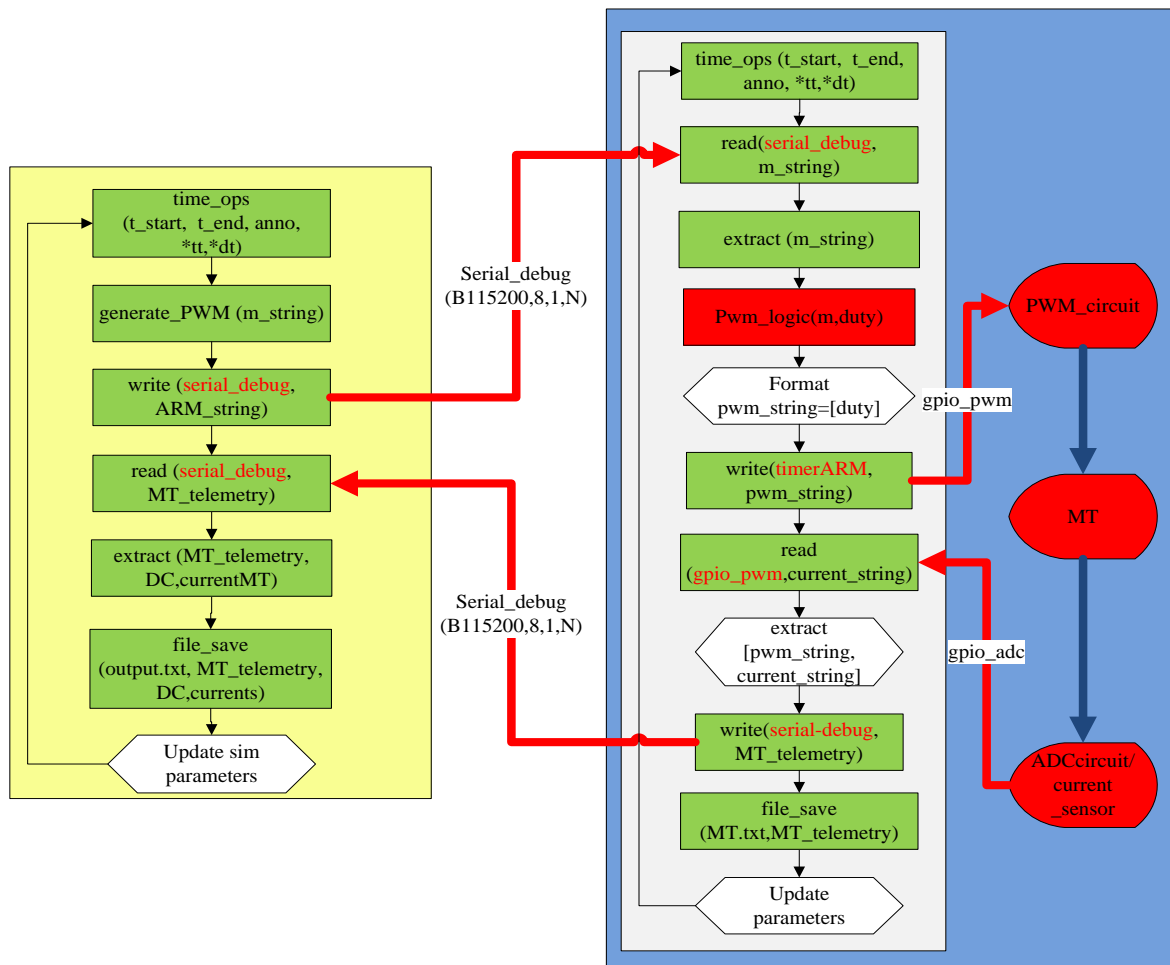


Figure 150: HIL simulations for PWM and current sensors calibration - models flow

```

HIL_PWM.c 23
Name      : Process.c

#include "headers.h"
#include "function_header.h"
#include "port_libs.h"

int main(int argc, char* argv[]) {
    /* *****
     *          INITIALIZATION
     * ***** */

    struct pipe *pipe_ptr;
    struct serial serial_debug, *serial_ptr;
    int n_pipes = 0, n_serials = 1, i = 0;

    if (argc < 2) {
        printf("Wrong number of arguments passed.");
        exit(EXIT_FAILURE);
    }

    i = 0;
    serial_ptr = &serial_debug;
    open_serials(argv[i + n_pipes + 2], serial_ptr);
    i++;

    /* *****
     *          VAR POOL
     * ***** */

    struct timeval end, start;
    double tt1, tstart,tt,dt,anno, Bmis[3] = {0}, wobm[3] = {0}, q[4] = {0};
    int currents[3] = {0}, DC[3] = {500,500,500}, dd, contatempo;
    float tf[6] = { 0 };
    char mt_telemetry[12] = { 0 };
    unsigned char ARM_string[25] = { 0 };

    system("rm -r -f /home/wsl/Dropbox/git/Simulatore/e-st@rII_test/Debug/LogFiles/");
    system("mkdir /home/wsl/Dropbox/git/Simulatore/e-st@rII_test/Debug/LogFiles/");
}

```



```

HIL_PWM.c
while (1) {
    gettimeofday(&start, NULL);

    time_ops(start, end, anno, tstart, &tt, &dt, tf);

    generate_PWM(ARM_string);

    dd = write(serial_debug.serial_fd, ARM_string, sizeof(ARM_string));
    if (dd == -1) {
        printf("Error in writing to port!");
        exit(EXIT_FAILURE);
    }
    printf("%d: bytes written: %d\n", getpid(), dd);

    dd = read(serial_debug.serial_fd, mt_telemetry, 12);
    if (dd == -1) {
        printf("Error in reading from port!");
        exit(EXIT_FAILURE);
    }
    printf("%d: bytes read: %d\n", getpid(), dd);

    extract(mt_telemetry, q, wobm, Bmis, currents, dd);

    file_save(&contatempo, mt_telemetry, DC, currents, tf, &tt1);
}
}

```

Figure 151: HIL simulations for PWM and current sensors calibration . C++ code skeleton

The picture in Figure 152 shows the arrangement of the test, using the QM ADCS board. The picture on the left provides a global view: there are the Simulation Unit (the WS, rear in the picture) with the control console on the right, the power supplier (in the middle) and the oscilloscope on the left. The picture on the right shows the detail of the Test Object: the MT is on the left and it is connected to the QM board; on the left there is GSE Front-end (a MAX232 circuit), connected with the Simulation Unit.

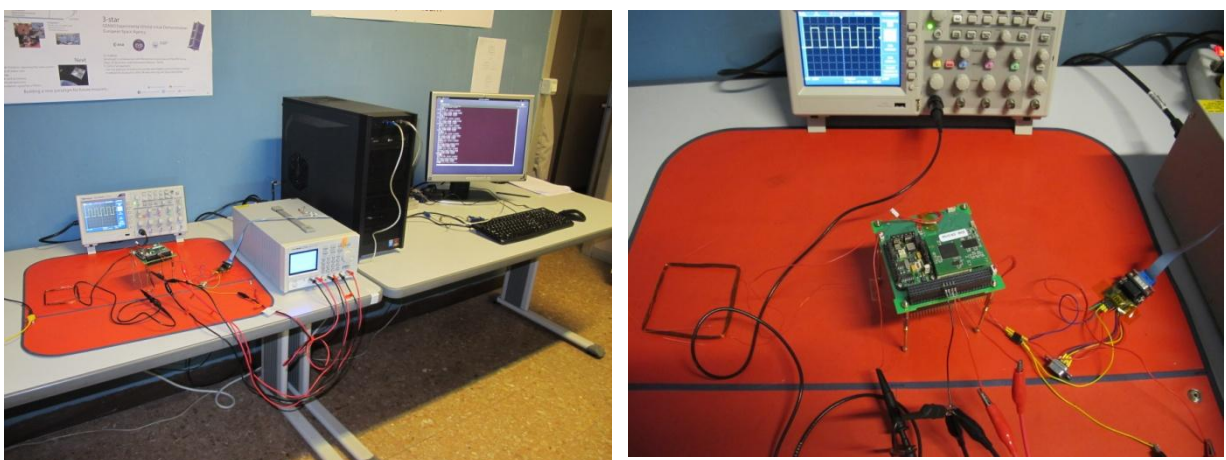


Figure 152: HIL simulations for PWM and current sensors calibration – picture of the arrangement for the test with FM

4.4.2.4.3 Results

The Figure 153 shows an approximately linear trend of the ADC wrt the duty cycle, except for the initial part (corresponding to low duty cycle values) where instead a saturation is present: this problem will be fixed on the ADCS board with changing resistors' values of the circuit. Figure 154 and Figure 155 report the relationships between the duty cycle values and the corresponding voltages and the measured currents flowing into the MTs connector.

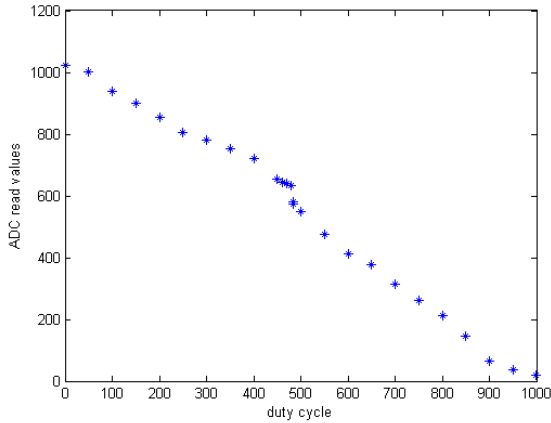


Figure 153: HIL simulation for PWM and current sensors calibration - PWM logic circuit calibration

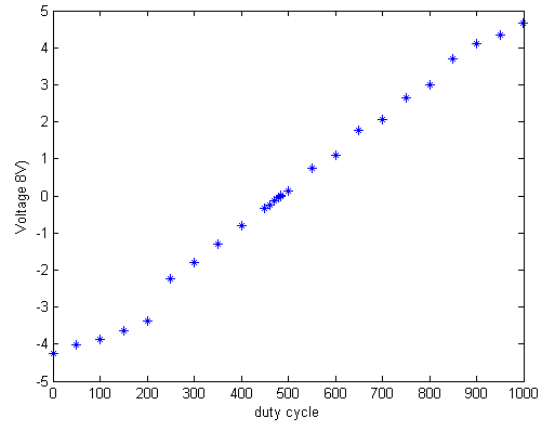


Figure 154: HIL simulation for PWM and current sensors calibrations - PWM tuning

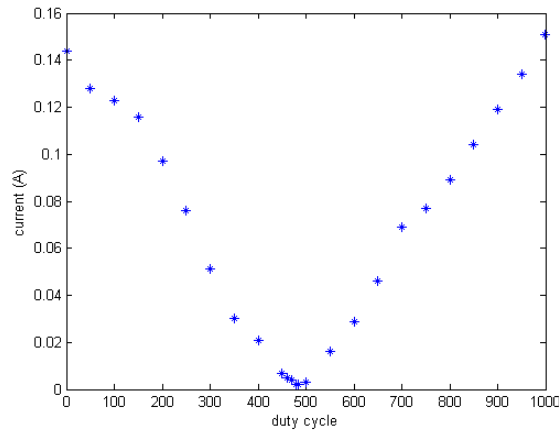


Figure 155: HIL simulation for PWM and current sensors calibration current sensor calibration

Figure 154 highlights that the center point is at 483: it means that when this value is set no current flowing within the MT. Moreover, the trend of the points underlines a quite linear relationship between duty cycle and output voltage. A quite linear relationship exists also between the duty cycle and the current flowing within the actuators (Figure 155).

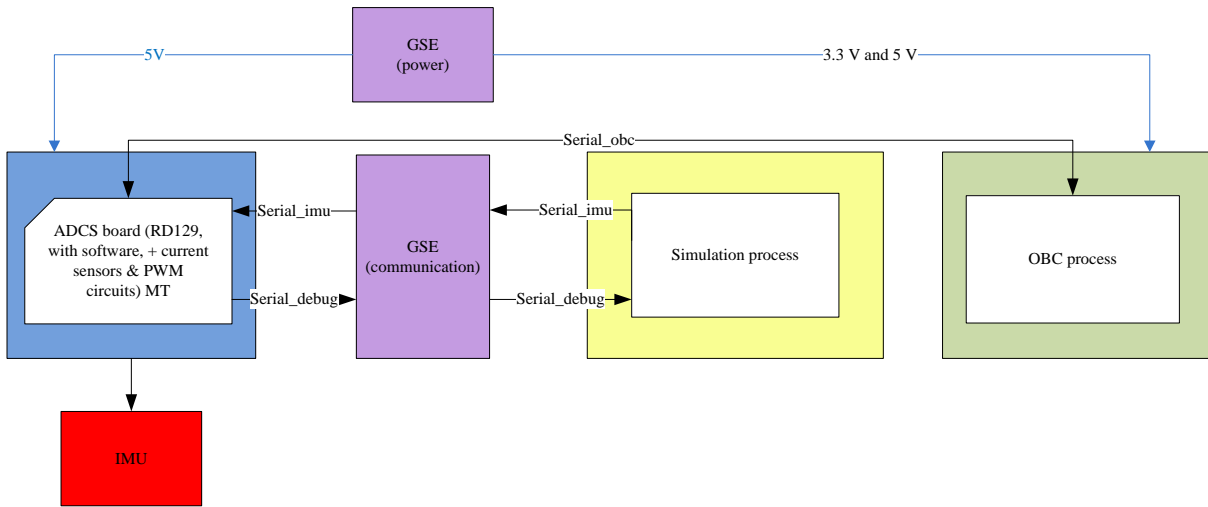
4.4.2.4.4 HIL simulation for complete A-ADCS verification

4.4.2.4.4.1 Objective

The main test objective is the verification of the full assembled and integrated ADCS: it means to verify the system functions, modes of operations and performances.

4.4.2.4.4.2 Setup and configurations

The architecture for the ADCS verifications is in Figure 156. The *simulation process* passes the IMU string through the *serial_imu* and receives the PWM commands on the *serial_debug*. A Front-end GSE is required to adapt the signals levels on both the links. The data exchange between ADCS board and OBC board (USART1) is guaranteed by a serial link. The ADCS microprocessor sends the PWM commands to MT setting the values on the RD129 *timer* channels' outputs; the signals are amplified in specific electronics circuits. The ADCS and OBC boards are powered on three different voltages (3.3V, 5V and [7-8,2]V) by a dedicated GSE.



```
setsimarch_COMPLETE_HIL_ADACS.txt x
#### - ARM
1
tempistica 1
priority 1
path /home/ws1/Dropbox/git/Simulatore/ARM/HIL_ADACS_COMPLETE/Release/HIL_ADACS_Process|
serial serial_obc /dev/ttySAC0 B115200
serial serial_debug /dev/ttySAC1 B115200
serial serial_imu /dev/ttySAC2 B115200

####
2
tempistica 1
priority 1
path /home/ws1/Dropbox/git/Simulatore/Sim/HIL_ADACS_COMPLETE/SIM/Release/HIL_ADACS_COMPLETE
serial serial_debug /dev/ttyUSB0 B115200
serial serial_imu /dev/ttyUSB1 B115200
```

Figure 156: HIL simulations for complete A-ADCS verification – processes settings

Figure 157 shows the “models flow” of the three processors involved in this final simulation campaign on A-ADCS at subsystem level. In the figure only the OBC functions necessary for interacts with the ADCS are reported: they are the communication of “O2A” strings and “A2O” strings on the *serial_obc* link. The ADCS processor (RD129) executes the software already described in the

paragraph 4.4.2.2: it reads the O2A data and verifies what mode of operations shall be setup and extracts other telemetry of interest. If no determination and control modes (ADCS-MODE 1 and ADCS-MODE 2) are set, the execution flow jumps to the end of the loop where the A2O string with the subsystem telemetry and status is formatted and sent to OBC. If the ADCS-MODE1 is activated the measurements of gyroscopes and magnetometers are read and used in the EKF and q-method algorithms to estimate the satellite's attitude (and the angular rate). If the ADCS-MODE 2 is activated, the control laws and the commands for MT are put in the loop. Moreover, the ADCS process handles and manages the received command from the OBC (and from the GS): they are the:

- “new desired attitude “
- “update orbit parameters”
- “update UTC time”

Accordingly to the active mode, the ADCS software executes the algorithms for attitude determination and the control laws.

The simulation process shall manage time and synchronization, reproduce the on orbit behavior of the satellite using dedicated models chosen by the user: for this final simulation the most accurate and with highest level of fidelity models have been selected both for equipment and for environmental and spacecraft motion virtual models. For example, the IMU models is characterized with the bias, misalignment, noise, and scale factor stability of the real sensor and taking into account the temperature changes. The *IMU_string* is transmitted (through the *serial_imu*) from the simulation process to the IMU-connector on the ADCS board. It receives also the PWM duty cycle commands calculated from A-ADCS and simulates the MT behavior. Finally, all the variables values and trends are saved in *output.txt* file.

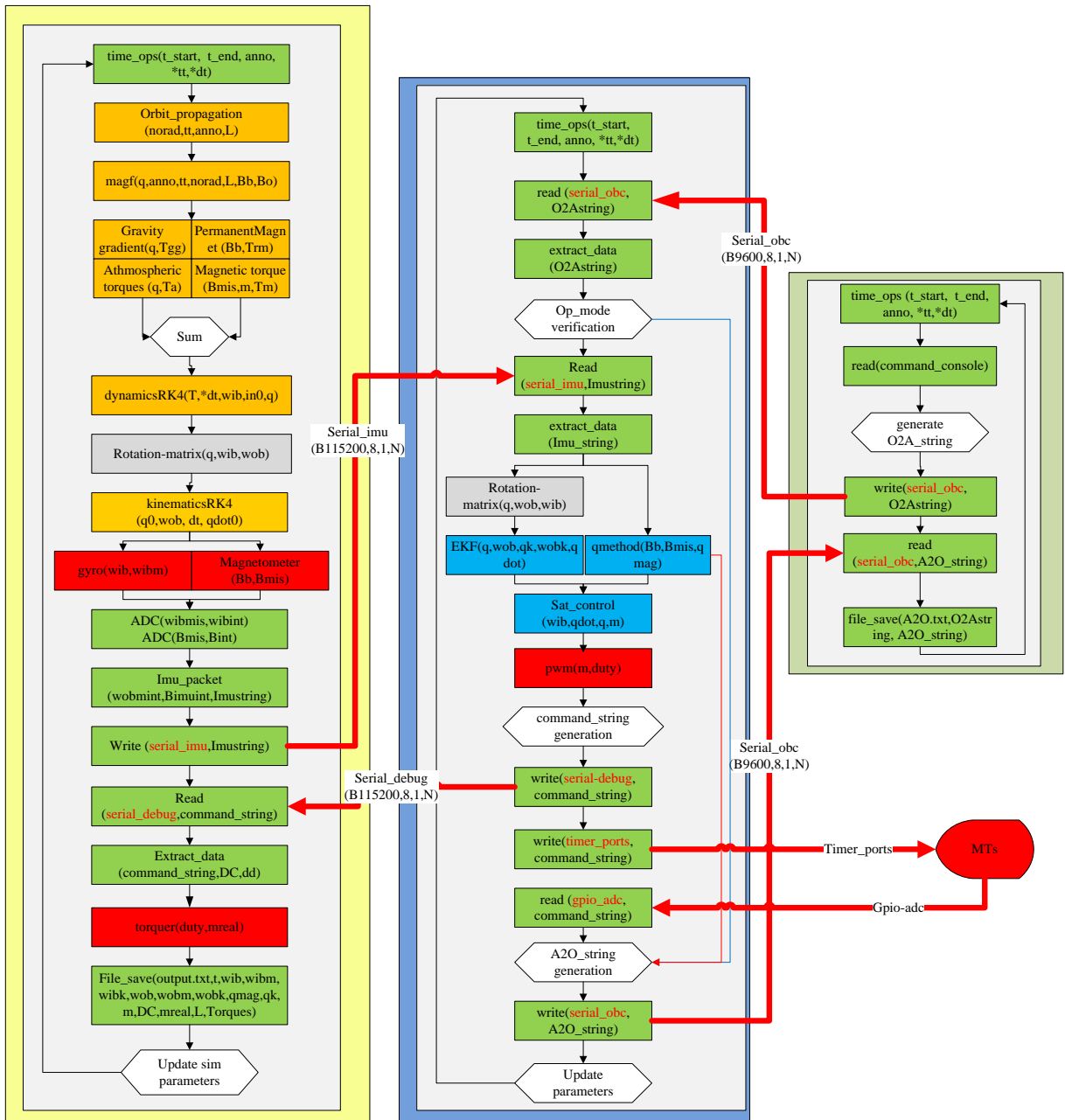


Figure 157: HIL simulation for complete A-ADCS verification - "models flow"

Figure 158 shows the C++ code for initialize the simulation: C++ libraries and databases files and all the variables and constants are properly defined.

```

HIL_ADCS.c
Name : Process.c

#include "headers.h"
#include "function_header.h"
#include "port_libs.h"

int main(int argc, char* argv[]) {

    /* *****
     *
     *          INITIALIZATION
     *
     * ***** */

    struct pipe *pipe_ptr;
    struct serial serial_imu, serial_debug, serial_obc, *serial_ptr;
    int n_pipes = 0, n_serials = 2, i = 0;

    if (argc < 2) {
        printf("Wrong number of arguments passed.");
        exit(EXIT_FAILURE);
    }

    i = 0;
    serial_ptr = &serial_imu;
    open_serials(argv[i + n_pipes + 2], serial_ptr);
    i++;

    serial_ptr = &serial_debug;
    open_serials(argv[i + n_pipes + 2], serial_ptr);
    i++;

    /* *****
     *
     *          VAR POOL
     *
     * ***** */

    struct timeval end, start;
    double wib[3] = { 0.6, 0.6, 0.05 }, norad[8] = { 98, 0, 0, 0, 0, 0, 0.0010, 0,
    0 }, tt = 0, anno = 0, B[3] = { 0.00001, 0.00001, 0.00001 }, I[6] =
    { 0 }, L[3] = { 0, 0, 600000 }, T[6] = { 20, 20, 20, 20, 20, 20 },
    dt = 0.5, m[3] = { 0, 0, 0 }, q[4] = { 0, 0, 0, 1 }, qc[4] = { 0, 0,
    0, 1 }, wibm[3] = { 0.6, 0.6, 0.05 }, wobm[3] = { 0.6, 0.6,
    0.05 }, in0[3] = { 0 }, qdot[4] = { 0 }, Torques[3] = { 0 },
    Tgg[3] = { 0 }, Ta[3] = { 0 }, Trm[3] = { 0 }, Tm[3] = { 0 },
    Bmis[3] = { 0 }, wob[3] = { 0 }, tt1, tstart;
    float tf[6] = { 0 };
    char command_string[150] = { 0 };
    unsigned char imu_string[25] = { 0 }, pwm_hil[6] = { 0 };
    int DC[3] = { 0, 0, 0 }, DCs[3] = { 0, 0, 0 }, contatempo, wibmint[3] = { 0,
    0, 0 }, dd = 0, ctrstab[2] = { 0, 0 }, wobmint[3] = { 0, 0, 0 },
    Bimuint[3] = { 0, 0, 0 };

    system("rm -r -f /home/wsl/Dropbox/git/Simulatore/e-st@rII_test/Debug/LogFiles/");
    system("mkdir /home/wsl/Dropbox/git/Simulatore/e-st@rII_test/Debug/LogFiles/");
}

```

Figure 158: HIL simulation for complete A-ADCS verification: C++ code generated skeleton (I)

```

HIL_ADACS.c
while (1) {
    gettimeofday(&start, NULL);
    time_ops(start, end, anno, tstart, &tt, &dt, tf);
    orbit_propagation(norad, anno, tt, L);
    magf(q, anno, tt, norad, B, L);
    gravity_gradient(q, Tgg);
    atmospheric_torque(q, Ta);
    permanent_magnet(B, Trm);
    magnetic_torque(m, Bmis, Tm);
    sum(Tgg, Ta, Trm, Tm, Torques);
    dynamics_RK4(Torques, dt, wib, in0, q);
    Rotation_matrix(wib, q, wob);
    kinematics(q, wob, dt, qdot);
    gyro(wob, wobm);
    imu(wobm, wobmint);
    magnetometer(B, Bmis);
    Bimu2int(Bmis, Bimuint);
    imu_packet(wobmint, Bimuint, imu_string);
    dd = write(serial_imu.serial_fd, imu_string, sizeof(imu_string));
    if (dd == -1) {
        printf("Error in writing to port!");
        exit(EXIT_FAILURE);
    }
    printf("%d: bytes written: %d\n", getpid(), dd);
    dd = read(serial_debug.serial_fd, command_string, 150);
    if (dd == -1) {
        printf("Error in reading from port!");
        exit(EXIT_FAILURE);
    }
    printf("%d: bytes read: %d\n", getpid(), dd);
    extract(command_string, DCs, dd);
    torquers(DCs, m);
    file_save(&contatempo, wib, wibm, wob, B, Bmis, tt, L, m, DCs, tf, &tt1,
            Torques);
}

```

Figure 159: HIL simulation for complete A-ADCS verification: C++ code generated skeleton (II)

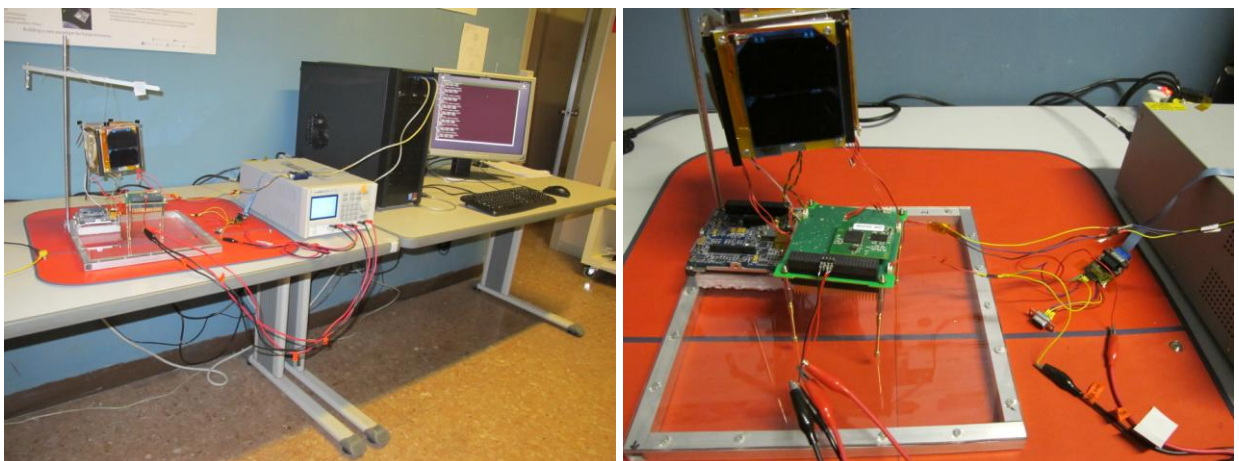


Figure 160: HIL simulation for complete A-ADCS verification - arrangement for the test with QM

Figure 160 shows the test setup arrangement. On the left, there is an overview of the test bench with Simulation Unit and Control Console, the power supplier and the test object. On the right, the details

of the test object is highlighted: the ADCS board is connected to the power supplier output to receive the regulated power on the main bus pins through the 104-pin connector.

4.4.2.4.4.3 Results

An extensive simulation campaign has been performed in order to verify the operational, functional and performances requirements of the ADCS before it was integrated with the other subsystems. The first simulations session fail mainly because the arrangement was not suitable and also because the communications do not result not sufficiently synchronize such that the real time operations were not guaranteed.

Hereafter, the results from the final session are shown: Figure 161 plots the body angular velocities wrt the orbital frame and the Figure 162 graphs the four components of the quaternion that represent the satellite attitude. The figures show that the ADCS is able to reach the determination and control attitude: it commands the actuators MT the initial angular velocities and pointing to the nadir the antenna. Consequently, the capabilities to activate and de-activate sensors and actuators have been successfully verified. From the communication point of view, the communication between the simulator and the ADCS are sufficiently stable. Although data anomalies are observed, from a minimum of 15 (best case) to a maximum of 45 (worst case) packets are corrupted as shows by peaks and discontinuities on the graphs, the ADCS has completed successfully the mission. In particular, these extrinsic performances have been obtained:

- $AKE(q)=[0.02511 \ 0.077 \ 0.0801 \ 0.005]$ NB. After a post-processing activities made o eliminate/recovery the communication error.
- $RKE(q)= [-0.0154 \ -0.0373 \ 0.0223 \ 0.0040]$ NB. wrt the last simulation acquisition
- $APE(q)= [0.2232 \ 0.3000 \ 0.2030 \ 1.0465]$ after the detumbling and acquisition phases
- $MPE(q)= [-0.1150 \ -0.0823 \ -0.0927 \ 0.0839]$
- $RPE(q)= [-0.0258 \ -0.0402 \ -0.0066 \ 0.0045]$ NB. wrt the last simulation acquisition

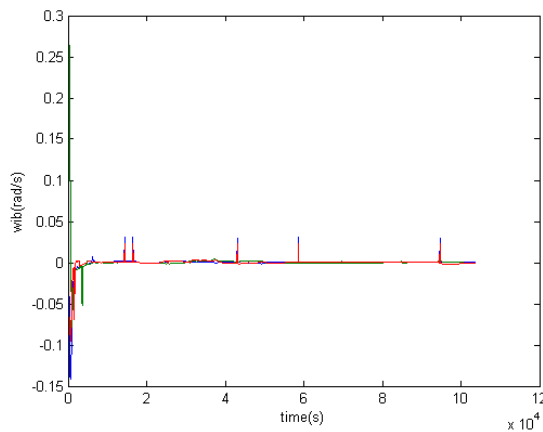


Figure 161: HIL simulation for complete A-ADCS verification - body angular velocity

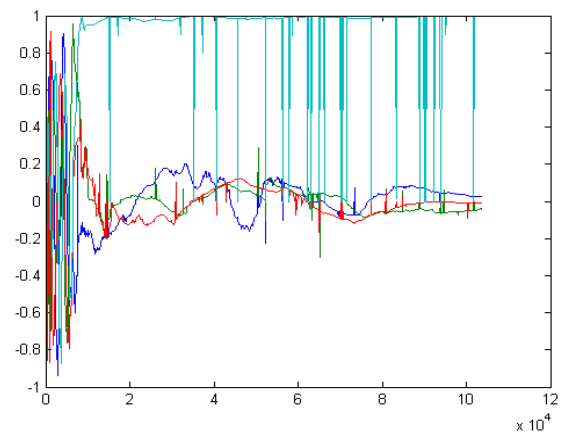


Figure 162: HIL simulation for complete A-ADCS verification – attitude

4.4.2.4.5 Comparison between AIL and HIL simulations results

The simulation results have been compared with the AIL simulations with the same initial conditions and general setup. Figure 163, Figure 164, and Figure 165 show the comparison among angular velocities and Figure 166, Figure 167, Figure 168, and Figure 169 the comparison among the quaternion values.

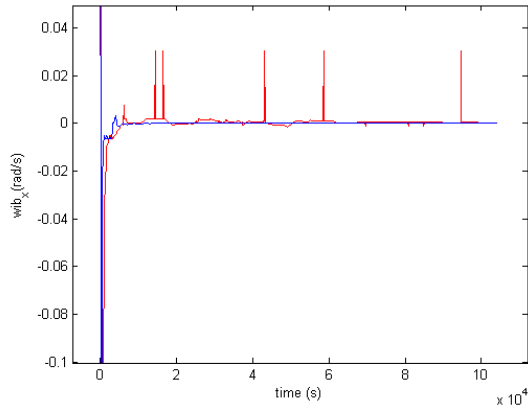


Figure 163: Body angular velocity wrt orbital frame (axis x), AIL(blue) vs. HIL(red)

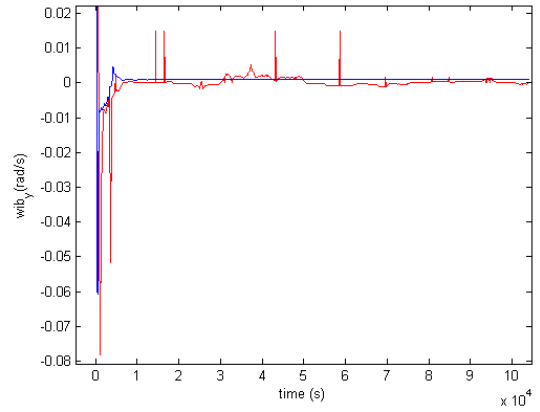


Figure 164: Body angular velocity wrt orbital frame (axis y), AIL(blue) vs. HIL(red)

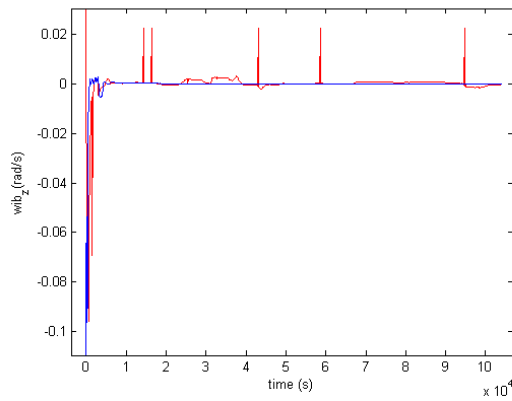


Figure 165: Body angular velocity wrt orbital frame (axis z), AIL(blue) vs. HIL(red)

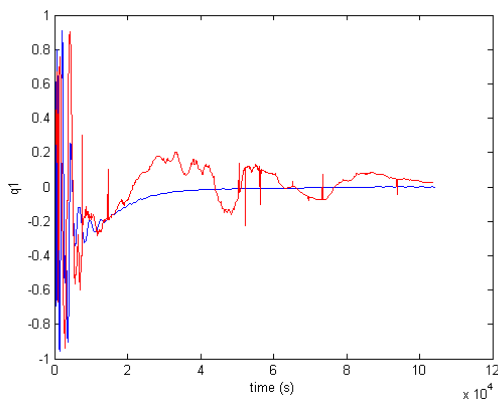


Figure 166: Attitude (component 1 quaternion vector part), AIL(blue) vs. HIL(red)

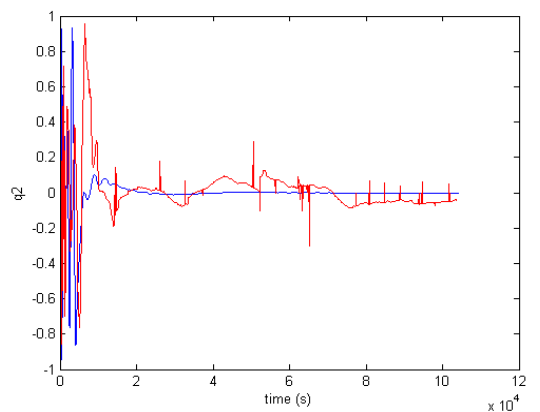


Figure 167: Attitude (component 2 quaternion vector part), AIL(blue) vs. HIL(red)

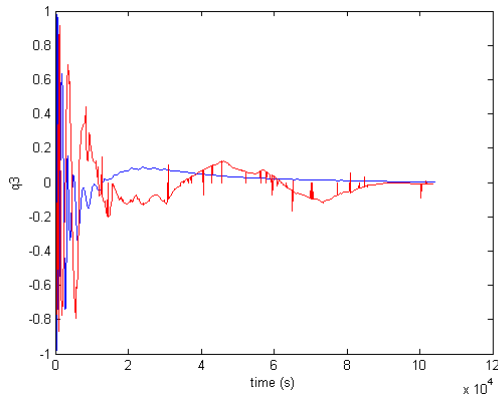


Figure 168: Attitude(component 3 quaternion vector part), AIL(blue) vs. HIL(red)

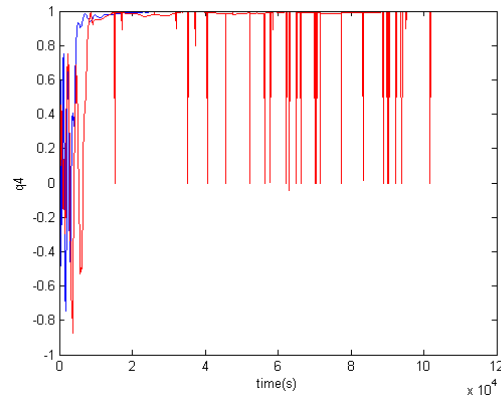


Figure 169: Attitude (component 4 quaternion vector part), AIL(blue) vs. HIL(red)

Apart for the communication errors, the AIL solutions tend to converge quickly than HIL: this is because not all the HW internal dynamics can be reproduced with the virtual models and they occur during a real time test. Moreover, the communication errors and transient misbehavior (e.g. the software reboot) not present in AIL actually affect the work of the hardware and software. Finally, Figure 170 shows the current consumption due to the actuators for complete the detumbling and the stabilization phase. The total consumption for attitude control is about 2.45 W.

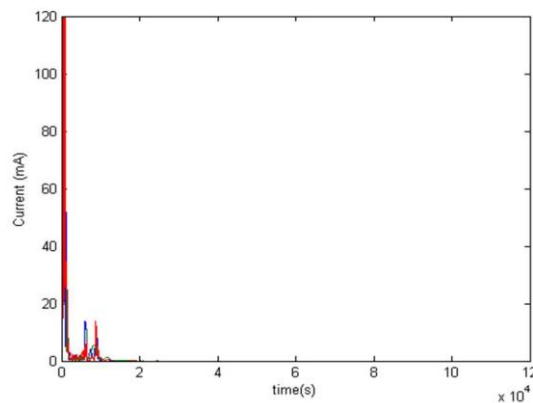


Figure 170: Current flowing within the MTs

4.4.2.5 HIL – integration and qualification phases at the system level

4.4.2.5.1 HIL simulation for complete e-st@r verification

4.4.2.5.1.1 Objectives

The test aims at verifying the complete A-ADCS system (and more, in general, the entire satellite) capabilities.

4.4.2.5.1.2 Configuration and setup

This section describes the StarSim configuration for the complete HIL simulation and test campaign carried out on the e-st@r CubeSats. The Test Object under verification at this stage is the whole satellite. Physical models of all the subsystems are in the loop in order to verify all functional and operational requirements, as Table 37.

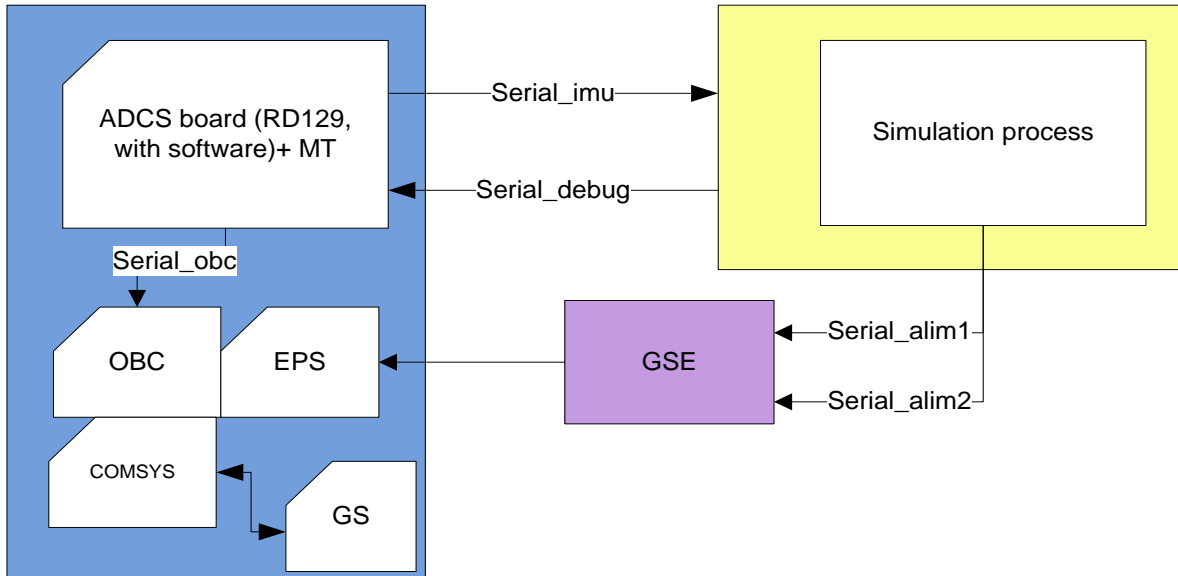
Description	Verification close-out
The CubeSat shall automatically deploy the antenna	Verified
The CubeSat shall send telemetry to GCS	Verified
The CubeSat shall receive commands from GCS	Verified
The CubeSat shall send telemetry packets every 120 s	Verified
Telemetry packets shall be formatted according to KISS AX.25 protocol	Verified
Telemetry shall be stored on board a non volatile memory (SD card)	Verified
The CubeSat shall execute commands received from GCS	Verified
The current consumption @5V shall be less than 150 mA in nominal mode of operation (mean value)	Verified
The current consumption @3.3V shall be less than 180 mA in nominal mode of operation (mean value)	Verified
The current consumption @5V shall be less than 200 mA in nominal mode of operation (peak value)	Verified
The current consumption @3.3V shall be less than 230 mA in nominal mode of operation (peak value)	Verified
Battery shall be charged in less than 10 hours	Verified
CubeSat shall determine its attitude	Verified
CubeSat shall control its attitude	Verified
CubeSat shall be able to maintain its attitude	Verified
CubeSat shall be able to change its attitude upon command from GCS	Verified
The pointing accuracy shall be at least 5°	Verified
The end of the detumbling phase, the angular velocities shall not exceed 0.005 rad/s for each axis	Verified
Current consumption of each MT shall be than 150 mA	Verified
Antenna deployment shall create any debris	Verified
Antenna shall wait to deploy a minimum of 30 minutes after the CubeSat deployment switch (DS) is activated from P-POD ejection	Verified
Transmitter shall wait to transmit a minimum of 30 minutes after the CubeSat's DS is activated from P-POD ejection	Verified
The CubeSat shall be able to receive a transmitter shutdown command as per FCC regulation	Verified
The CubeSat shall be able to switch from one operation mode to another operation mode upon command from GCS	Verified
No electronics shall be active during launch to prevent any electrical or RF	Verified
All systems shall be turned off by the DS, including real time clocks	Verified
CubeSats with batteries shall be fully deactivated or launch with discharge batteries	Verified

Table 37: Functional and operational requirements verified by complete HIL simulation

The hardware interfaces of the HIL test bench are listed hereafter:

- *Serial_debug* port connects the Simulation Unit to RD129 USART2 debug port on the A-ADCS board using a GSE Front-end based on the MAX232 chip. The A-ADCS board returns the Pulse Width Modulation (PWM) commands to the simulation unit. Through this channel, the operator controls the A-ADCS's behavior in real time.
- *Serial_imu* port connects the simulation unit to the IMU on the A-ADCS board. Simulated IMU's raw data are transmitted to the A-ADCS board in terms of angular velocities and local EMF, formatted as specified in the IMU data-sheet.
- *Serial_alim1* connects the Simulation Unit to power GSE (power pack N1). The simulation software conveys solar panels current and voltage values to set power pack N1 in order to simulate solar panels +x, +y, -y. The three power pack's outputs are connected directly to the six-pin connectors on the PCDU.

- *Serial_alim2* connects the simulation unit to power GSE (power pack N2). The simulation software conveys solar panels current and voltage values to power pack N2 in order to simulate solar panels +z and -z. Two out of the three power pack’s outputs are connected directly to the six-pin connectors on the PCDU.
- *Serial_abc* connects the ADCS and OBC systems through a serial cable or, in the QM e FM units, through the 104-pin connector of the e-st@r bus. Actually, this communication channel does not involve the SU but it is entirely part of the Test Object.



```
setsimarch_COMPLETE_HIL.txt x
####
1
tempistica 1
priority 1
path /home/ws1/Dropbox/git/Simulatore/Sim/HIL_COMPLETE/SIM/Release/HIL_COMPLETE
serial serial_debug /dev/ttyUSB0 B115200
serial serial_imu /dev/ttyUSB1 B115200
serial serial_ps1 /dev/ttyUSB3 B9600
serial serial_ps2 /dev/ttyUSB4 B9600

#### - ARM
2
tempistica 1
priority 1
path /home/ws1/Dropbox/git/Simulatore/ARM/HIL_COMPLETE/Release/HIL_ADCS_Process
serial serial_debug /dev/ttySAC0 B115200
serial serial_imu /dev/ttySAC1 B115200
```

Figure 171: HIL simulation for complete e-st@r verification – processes and interfaces settings

Figure 172 shows the details of the configuration through the “models flow” for the *simulation process* and the ADCS software. The simulation unit also includes electrical emulation of sensors and actuators. The output value of each sensor is computed by the simulator and it is read by the ADCS board under test. In the same way, the ADCS software commands the actuators by means of proper control signals and it sends back to the simulator the control values.

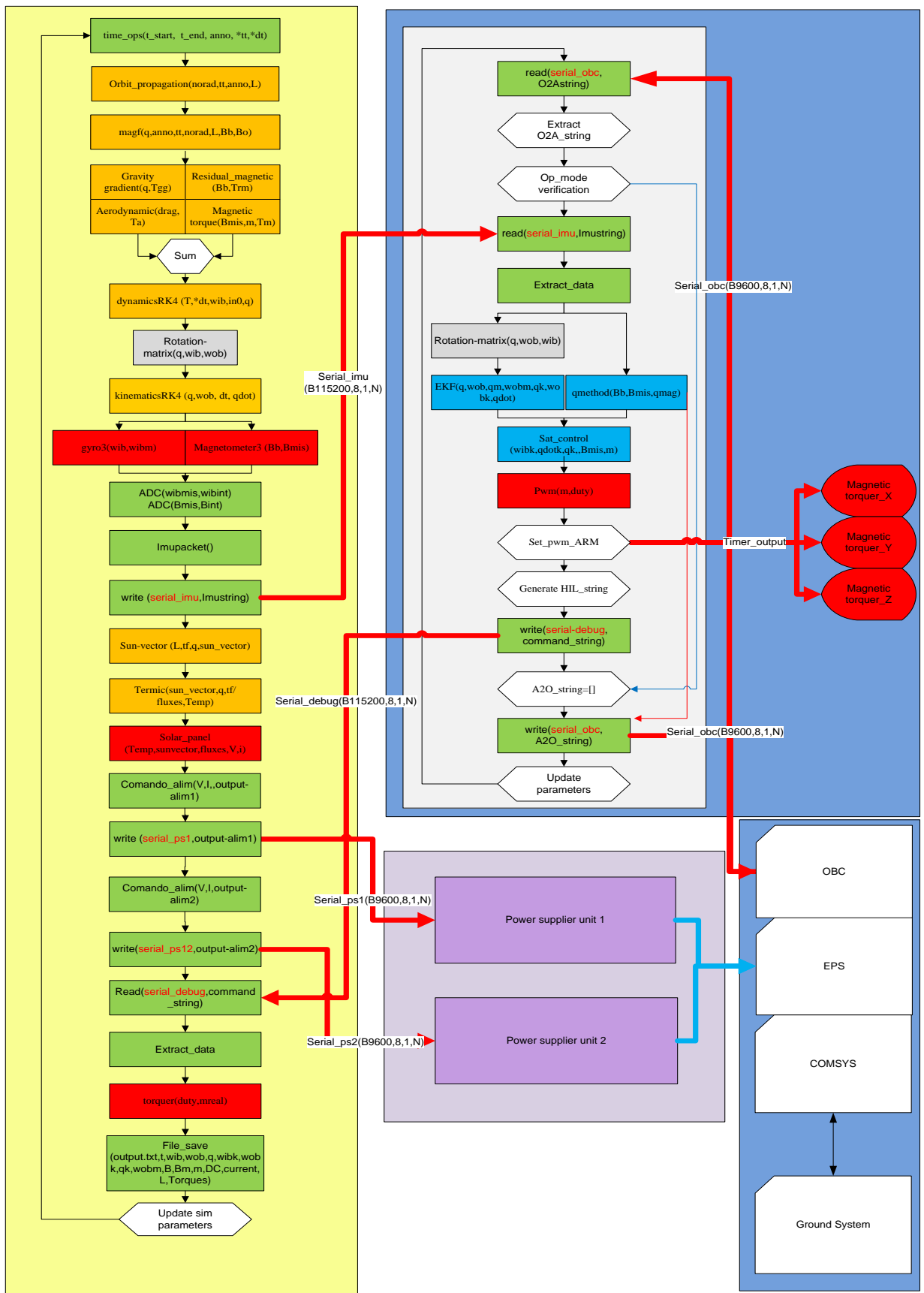


Figure 172: HIL simulation for complete e-st@r verification – models flows

The simulated satellite's equipment are:

1. Solar panels and EPS's temperature sensors. Their models have been developed from data according to [29]. Using the simulated information of Sun position, satellite attitude, and thermal conditions, the voltage and current supplied by each panel are computed. These information are used to set the power units that are connected to the EPS board to power the CubeSat with the desired voltages and currents
2. IMU. The models of the gyroscopes and of the tri-axial magnetometer are included in the simulator [30]. These models are used in the attitude determination loop
3. MT. The models of the magnetic actuators [30] and the PWM drivers [31] are used to calculate the dipole moment that, together with the EMF simulated data, generates the control torque.

```

HIL_completo.c
int main(int argc, char* argv[]) {
    /* *****
     *
     *          INITIALIZATION
     *
     * ***** */

    struct pipe *pipe_ptr;
    struct serial_serial_imu, serial_debug, serial_ps1, serial_ps2, *serial_ptr;
    int n_pipes = 0, n_serials = 4, i = 0;

    if (argc < 2) {
        printf("Wrong number of arguments passed.");
        exit(EXIT_FAILURE);
    }

    i = 0;
    serial_ptr = &serial_imu;
    open_serials(argv[i + n_pipes + 2], serial_ptr);
    i++;

    serial_ptr = &serial_debug;
    open_serials(argv[i + n_pipes + 2], serial_ptr);
    i++;

    serial_ptr = &serial_ps1;
    open_serials(argv[i + n_pipes + 2], serial_ptr);
    i++;

    serial_ptr = &serial_ps2;
    open_serials(argv[i + n_pipes + 2], serial_ptr);
    i++;

    /* *****
     *
     *          VAR POOL
     *
     * ***** */

    struct timeval end, start;
    double wib[3] = { 0.2, 0.2, 0.02 }, norad[8] = { 98, 0, 0, 0, 0, 0.0010, 0,
    0 }, tt = 0, anno = 0, B[3] = { 0.00001, 0.00001, 0.00001 }, I[6] =
    { 0 }, L[3] = { 0, 0, 600000 },
    dt = 0.5, m[3] = { 0, 0, 0 }, q[4] = { 0, 0, 0, 1 }, wibm[3] = { 0.2, 0.2, 0.02 },
    wobm[3] = { 0.2, 0.2, 0.02 }, in0[3] = { 0 }, qdot[4] = { 0 }, Torques[3] = { 0 },
    Tgg[3] = { 0 }, Ta[3] = { 0 }, Trm[3] = { 0 }, Tm[3] = { 0 }, V[6] = {0},
    Bmis[3] = { 0 }, wob[3] = { 0 }, ttl, tstart, fluxes[3] = {0}, temperatures[6] = {0};
    float tf[6] = { 0 };
    char command_string[150] = { 0 }, ps1_command[81] = {0}, ps2_command[81] = {0};
    unsigned char imu_string[25] = { 0 };
    int DC[3] = { 0, 0, 0 }, DCS[3] = { 0, 0, 0 }, contatempo, dd = 0, wobmint[3] = { 0, 0, 0 },
    Bimuint[3] = { 0, 0, 0 };

    system("rm -r -f /home/wsl/Dropbox/git/Simulatore/e-st@rII_test/Debug/LogFiles/");
    system("mkdir /home/wsl/Dropbox/git/Simulatore/e-st@rII_test/Debug/LogFiles/");

```

Figure 173: HIL simulation for complete e-st@r verification – C++ generated code skeleton (initialization)

The orbital conditions are simulated through the models of: 1) the orbit by a Simplified General Perturbation propagator (SGP4); 2) the rotational dynamics and kinematics of the satellite; 3) the thermal flows hitting the satellite [32]; and 4) the EMF [33]. The orbital position is necessary for obtaining the local sun vector and the local magnetic field. The sun vector is used to calculate thermal flows and temperatures on solar panels. All these models are implemented in the simulation unit of the HIL simulator. Sensors and actuators of the CubeSat are electrically connected to the power bus, but only for verification of total power consumption. They are not used for sensing data or for generating torques. Figure 174 shows the core of the generated code.

The final HIL configuration for the entire satellite requires a deeper discussion.

After the simulation is initialized, PWM commands are calculated in terms of duty cycles. These commands are sent to the simulation unit through *serial_debug* channel. The simulation software calculates the applied voltages, and the currents flowing into the MTs. In the end, the generated dipole moment is determined.

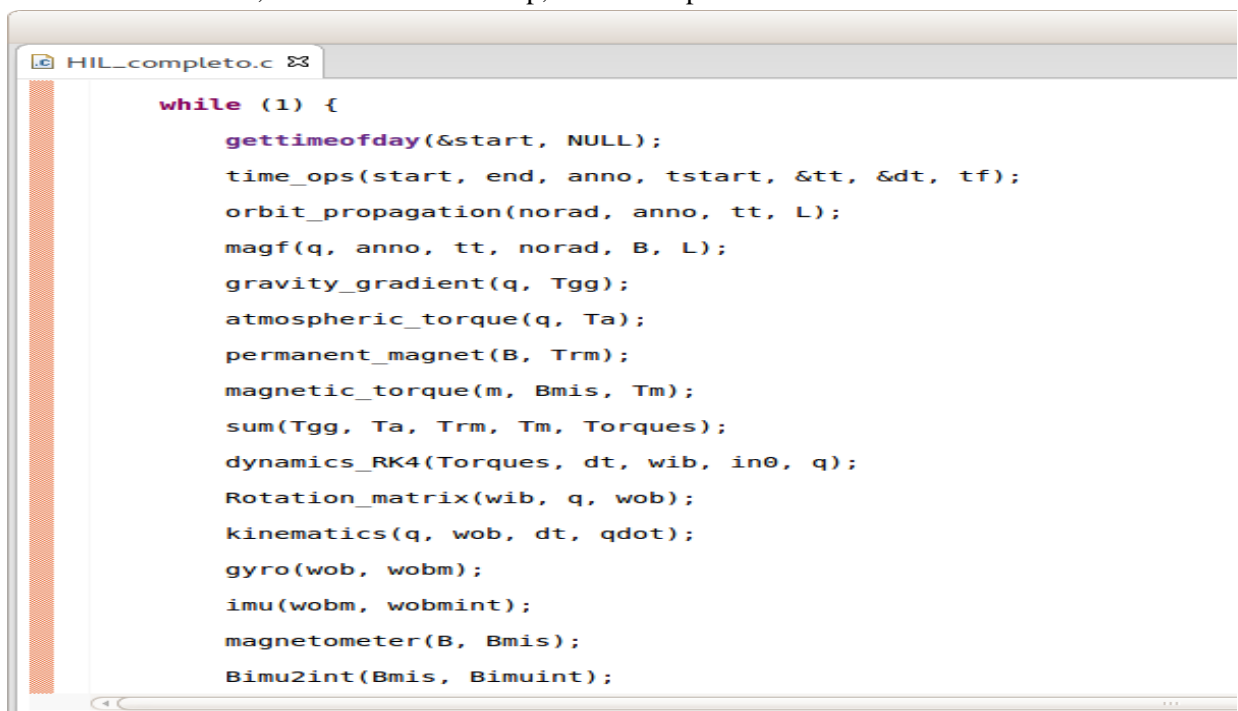
The dipole moment together with the Earth's magnetic field (measured by the modeled magnetometer) generates the value of the magnetic control torque. Using the dynamics and kinematics models of the CubeSat it is possible to obtain the values of the angular velocities around the satellite's body axes with respect to the inertial reference frame.

The angular velocities values (measured by gyros) and EMF values (measured by magnetometer) are converted in data strings in order to emulate the actual IMU's behavior; they are finally transmitted to the A-ADCS board through *serial_imu* channel. The simulated measurements are used by the A-ADCS board to determine the attitude. Moreover, attitude, angular velocities and EMF values serve to control the attitude of the satellite computing the new PWM duty cycles and finally sending them back to the simulation unit.

The simulation software determines the heat fluxes on each face of the CubeSat and the temperatures on each solar panel. Values of heat fluxes and temperatures enter the solar cells model in order to get the power generated by solar panels in terms of voltage and current. These information are transmitted (through *serial_alim1* and *serial_alim2* channels) to the two 3-channels power packs which supply the EPS board.

Telemetry is also radio-transmitted by the CubeSat to the GCS where it can be visualized and saved for further analysis. The CubeSat may also receive commands from GCS during a simulation session.

Figure 173 reports the first part of the generated code in which all the variables and constants of the models are initialized, the interfaces are setup, and the output files are selected.



```
while (1) {
    gettimeofday(&start, NULL);
    time_ops(start, end, anno, tstart, &tt, &dt, tf);
    orbit_propagation(norad, anno, tt, L);
    magf(q, anno, tt, norad, B, L);
    gravity_gradient(q, Tgg);
    atmospheric_torque(q, Ta);
    permanent_magnet(B, Trm);
    magnetic_torque(m, Bmis, Tm);
    sum(Tgg, Ta, Trm, Tm, Torques);
    dynamics_RK4(Torques, dt, wib, in0, q);
    Rotation_matrix(wib, q, wob);
    kinematics(q, wob, dt, qdot);
    gyro(wob, wobm);
    imu(wobm, wobmint);
    magnetometer(B, Bmis);
    Bimu2int(Bmis, Bimuint);
}
```

```

HIL_completo.c
imu_packet(wobmint, Bimuint, imu_string);

dd = write(serial_imu.serial_fd, imu_string, sizeof(imu_string));
if (dd == -1) {
    printf("Error in writing to port!");
    exit(EXIT_FAILURE);
}
printf("%d: bytes written: %d\n", getpid(), dd);

sun_vector(L, tf, q, sun_vector);

termic(sun_vector, q, tf, fluxes, temperatures);

solar_panels(sun_vector, temperatures, fluxes, V, I);

comando_alimentatore1(V, I, ps1_command);

dd = write(serial_ps1.serial_fd, ps1_command, sizeof(ps1_command));
if (dd == -1) {
    printf("Error in writing to port!");
    exit(EXIT_FAILURE);
}
printf("%d: bytes written: %d\n", getpid(), dd);

comando_alimentatore2(V, I, ps2_command);

dd = write(serial_ps2.serial_fd, ps2_command, sizeof(ps2_command));
if (dd == -1) {
    printf("Error in writing to port!");
    exit(EXIT_FAILURE);
}
printf("%d: bytes written: %d\n", getpid(), dd);

dd = read(serial_debug.serial_fd, command_string, 150);
if (dd == -1) {
    printf("Error in reading from port!");
    exit(EXIT_FAILURE);
}
printf("%d: bytes read: %d\n", getpid(), dd);

extract(command_string, DC, dd);

torquers(DCs, m);

file_save(&contatempo, wib, wibm, wob, B, Bmis, tt, L, m, DCs, tf, &ttl,
V, I, temperatures, Torques);
}

```

Figure 174: HIL simulation for complete e-st@r verification – C++ generated code skeleton (II)

Figure 175 shows the arrangement of the test object, GSE and the simulator for the complete HIL test on the integrated system.

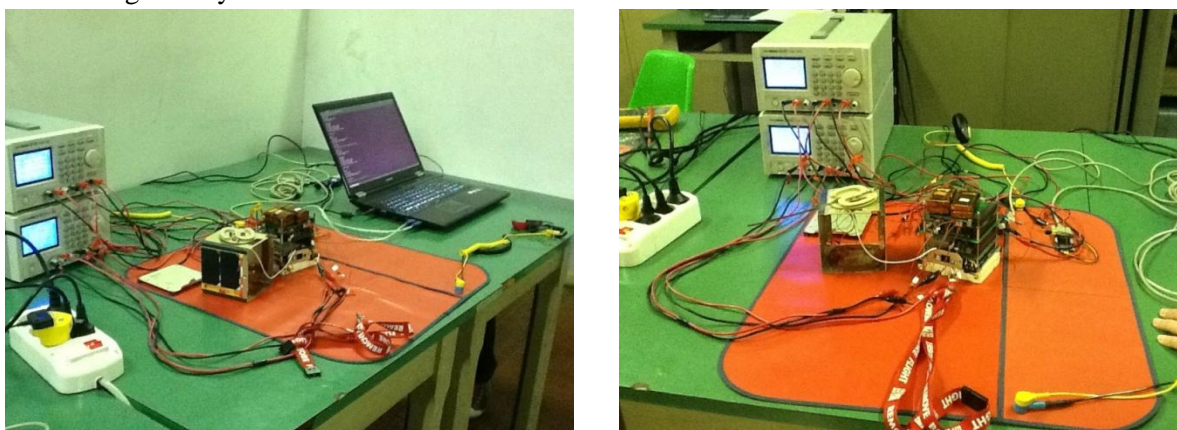


Figure 175: Complete HIL for e-st@r functional reqs verification – arrangement for the test with QM

Key parameters are defined before the simulation starts, such as the duration of the simulation (1 day), the time step (0.5 seconds), and the models to be used. CubeSat’s characteristics are also chosen and their values (peak or trends) are continuously monitored as they shall be verified against requirements. For e-st@r, the parameters are the EPS and ADCS telemetries and other system’s health and status data.

The analysis of simulation results is made from data collected through three independent sources: 1) GCS log files: updated every time that a radio communication between the satellite and the GCS occurs and a new packet, containing the telemetry data, is received; 2) Satellite SD card files: updated every 2 minutes; they contain the same telemetry strings received by the GCS but they are free of transmission errors; 3) Simulator log files: updated every 0.5 seconds; they hold all data about the environmental and system simulation and not only the telemetry data.

At the present moment, data analysis is performed off real time, following a procedure that consists of importing data from files, handling, converting and interpreting them. The most meaningful parameters are chosen and the respective graphs are plotted and evaluated.

4.4.2.5.1.3 Results

In this paragraph, the results of a simulation session are presented and discussed. The global objective of the campaign was the study of the satellite's behavior in the first day of in-orbit operations. The test aims at demonstrating that the CubeSat performs according to a reference mission as described hereafter. Mission and orbit data are given in Table 38.

Semi-major axis [km]	7359.46
Eccentricity	0.06664
Inclination [deg]	69.5
RAAN	0
Argument of perigee [deg]	130
Mean anomaly [deg]	0
Epoch	1 Jan 2012
Date of launch	13 Feb 2012
Launch site	Kourou, French Guyana
Launch Vehicle	Vega

Table 38: Mission and orbit data

The CubeSat is released into orbit at time T_0 . It enters the *detumbling* mode during which the A-ADCS reduces the angular velocities until threshold values are reached. The CubeSat remains in this mode of operations for about 2 hours. At $T_0 + 2$ hours: 1) the antenna deploys autonomously, 2) COMSYS is activated by OBC, and 3) the communication link from/to Earth is established. In the nominal mission, telemetry packets are sent to GCS every 2 minutes. According to the test plan, commands from the GCS are sent to the CubeSat to update mission parameters and to change modes of operations.

At the end of the simulation session, the log files are analyzed to evaluate the CubeSat's behavior, and to verify if the requirements are met. In the following graphs, most interesting parameters' trends are plotted from data in the GCS log file. The simulation has been running for 100000 seconds. After that period of time, the simulator did not control the process anymore but data were still collected at the GCS because the CubeSat was operative to perform additional off-line tests.

In Figure 176, battery-bus voltage, 5V-bus current, and 3.3V-bus current trends are shown. The battery bus voltage graph shows that the batteries are recharged during the orbits, reaching the maximum voltage value equal to 8.17 V in less than 10 hours (given a starting value of 7.80 V). Power consumption on the 5V-bus is almost constant. The peak in the plot is due to a wrong data acquisition; power consumption on the 3.3V-bus varies depending upon the use of OBC and A-ADCS boards.

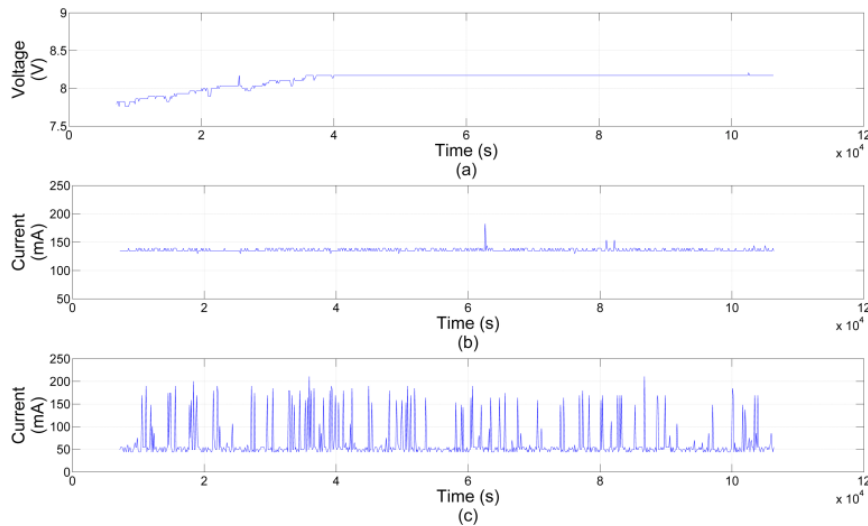


Figure 176: (a) Battery-bus voltage as a function of time, (b) 5V-bus current as a function of time, and (c) 3.3V-bus current as a function of time

In Figure 177, the trends of relevant parameters of one battery cell are shown. Each battery pack is constituted by two Li-Ion (Lithium Ions) cells. As far as voltage of cell is concerned, the graph confirms the trend seen in Figure 176 for the battery-bus voltage. The plot of the current shows consumption peaks which depend on the variable instantaneous operations of the subsystems. Since the test was performed in laboratory conditions, the temperature is almost constant and is equal to about 27 °C (which is higher than the room temperature, due to the fact that the battery heats up during operations, in particular during charge).

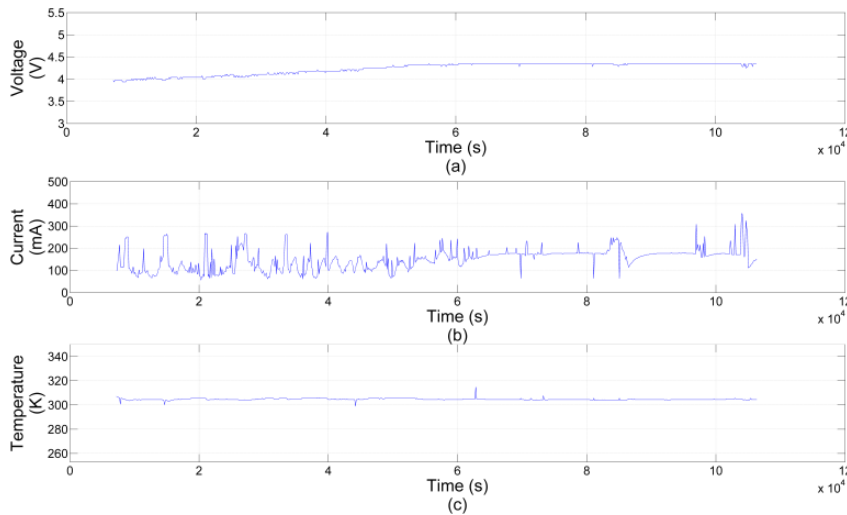


Figure 177: Telemetries of battery pack no. 1 as a function of time. (a) Voltage, (b) current, and (c) temperature of one cell of battery pack

Figure 178 illustrates the plots of solar panels voltages during the one-day mission, showing the sunlight/eclipse cycles. The graphs show that the satellite experiences, within the same day, both orbits with short-eclipse/long-sunlight periods and orbits with no eclipse. This is due to the combination of orbit geometry and launch date. It represents a transition condition toward a one-week period of full sunlight. During the simulated mission, the power consumption of the CubeSat is quite low, as only vital functions are executed. Moreover, Figure 178 shows very short-duration eclipses. For these reasons, we can see very light voltage drops in Figure 176(a). The anomalous values in the

solar panels' voltage trends beyond 10^5 seconds are due to the fact that the simulation was stopped and the power supply units were not controlled by the simulator.

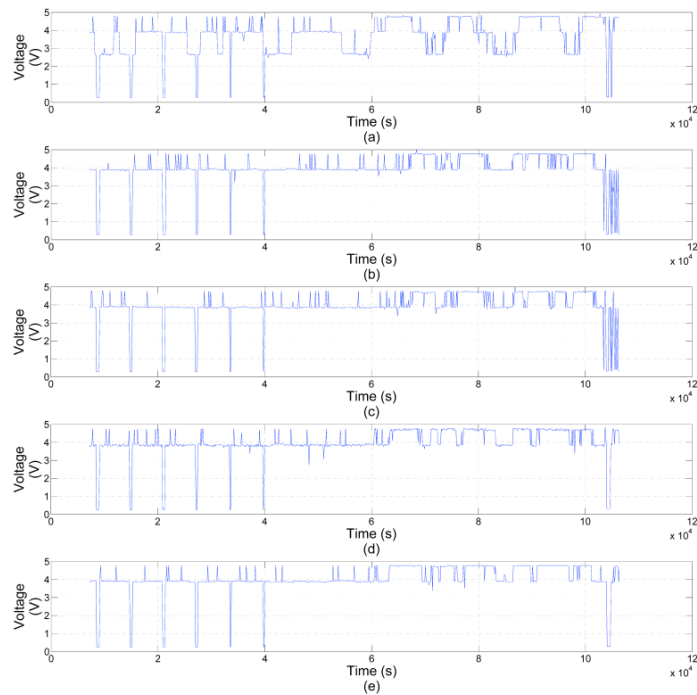


Figure 178: Solar panel voltages. (a) panel +x, (b) panel +y, (c) panel -y, (d) panel +z, and (e) panel -z. Panels are named after the perpendicular vector of the face they lie on, in body axes.

After the *detumbling* phase ends, the angular velocities of the CubeSat remain constant with respect to the inertial frame, therefore the satellite is not spinning anymore. In Figure 179, it is possible to notice that the angular velocity with respect to the orbital frame is close to 0. The trends of the angular velocities during the *detumbling* phase are not visible in Figure 179 because the graphs have been plotted from the GCS log file which does not record any data before the *detumbling* phase is completed. The peak in the ω_x plot is ascribable to an acquisition error.

Figure 180 shows that the desired attitude (antenna pointing to the ground, reference quaternion [0 0 0 1]) is achieved with the expected accuracy in the first day of the mission. The pointing error measured at the end of the test is less than 10 deg.

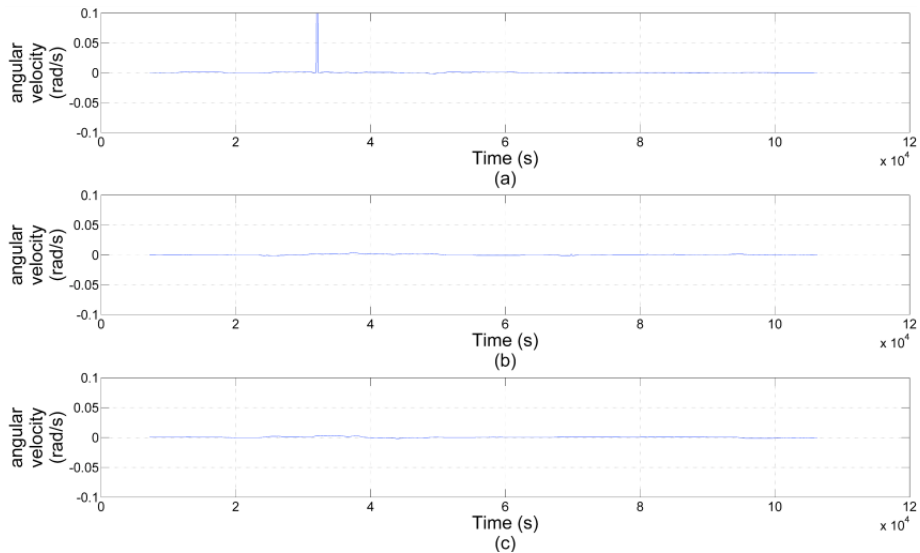


Figure 179: Angular velocities as a function of time. (a) angular velocity along x axis, (b) angular velocity along y axis, and (c) angular velocity along z axis

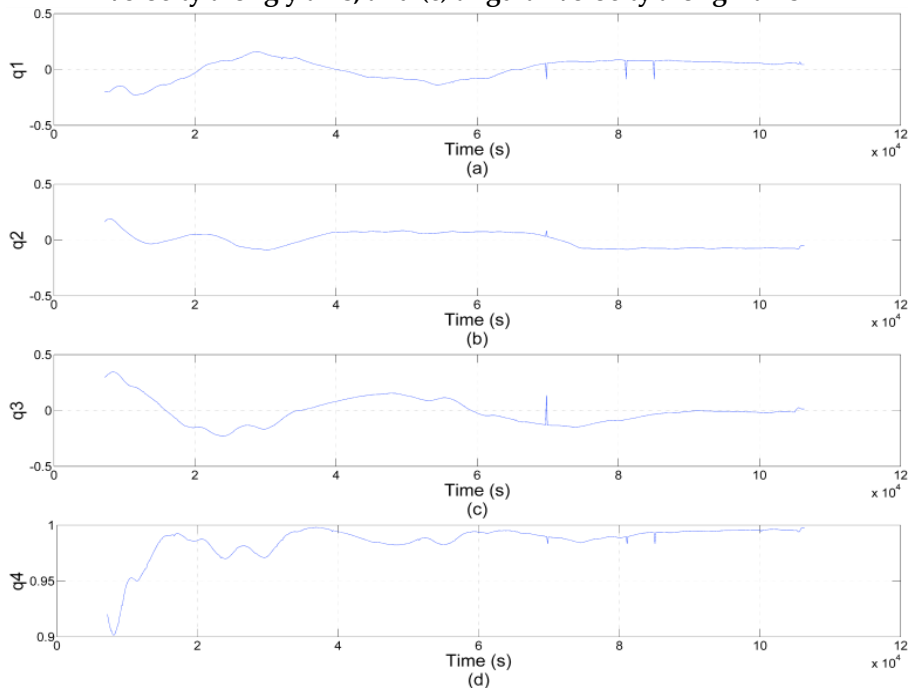


Figure 180: Quaternion. (A) First component of the quaternion vector part, (B), second component of the quaternion vector part (C) third component of the quaternion vector part, (D,) quaternion scalar part

Moreover, from all graphs, it is possible to see that no data are received at GCS during the first 2 hours after injection into orbit. This was one of the operational requirements to be verified. The set of requirements and constraints verified in the same HIL simulation session is reported in Table 37. In addition to the closeout of these requirements, other interesting information is obtained from the test:

1. No packet is lost (not saved and/or not received) in one-day span simulation
2. One system reset occurred over one-day span simulation, demonstrating the robustness of the software design

3 out of 720 packets contain no-coherent data (comparison between data saved on the onboard SD card and data recorded in the GCS log file). The data error rate (measured in terms of packets per day) is less than 5×10^{-3} .

4.4.3 Test case: lessons learned

The test case shows an application of the HIL simulation technique based on the simulator developed within the PhD program. The simulator has been developed to support the engineering team with a versatile tool which can be used throughout the design and development process of a space system. Thanks to its modular architecture, the simulator shows a good flexibility with respect to project phases and types of missions/systems. It may be tailored for different applications without major changes and it can handle simulation of several kinds of space missions and systems. Simulation models may be loaded in the simulator according to the specific design stage: models with different levels of detail and complexity are available and new models can be implemented and integrated in the simulator in the future. The type of mission of interest can also be chosen by setting the simulator in terms of specific mission parameters and systems involved in the simulation session.

At the present moment, only simple designs can be investigated through this platform, but it can be extended for testing more complex mission without re-designing the simulator's architecture. Some improvements can be done in order to enhance the simulator performance and accuracy, as well as to simplify the test bench and to reduce the time for test set up. Most common interfaces are already implemented in the simulator, but we are working to include other protocols. For example, the integration of the CAN bus interface could make the simulator suitable for testing pieces of hardware not supported at present. The addition of a robotic arm for improving attitude simulation capability is also planned. A key feature of the simulator is the use of freeware. However, well known commercial products have been used to "validate" the HIL simulations results wherever possible.

The HIL simulator has been used for the verification of the functional and operational requirements of the e-st@r-I CubeSat. A one-day span mission has been simulated during which the CubeSat demonstrated to be able to accomplish the required functions and to operate as expected.

The whole mission simulation has been carried out without relevant errors, showing that the HIL simulator can support the verification process of a CubeSat. A large number of requirements have been verified within a single simulation session, thus saving resources in terms of time for test set-up and running, man-hours and costs if compared to other testing techniques. Moreover, the test allowed following out the behavior of the equipment included in the simulation loop. For example, the actual power consumption of each piece of hardware has been measured and used to refine the power budget. The HIL testing results have been compared with AIL and SIL simulations to "validate" the methodology and the facility. These comparisons showed a good correlation between the different simulations, even if some differences have been noticed. For example, using the HIL facility helped to identify unexpected interactions between different devices during the integration of the CubeSat. The design has been reviewed to eliminate these potential failure causes using the information given by the HIL simulations.

In conclusion, it has been proven that HIL simulation can effectively support the AIV process of a CubeSat. Improving the verification activity of CubeSats is a key factor for the development of innovative CubeSat missions. It has been demonstrated that a simple HIL architecture can serve well the purpose, even if some limitations exist. The HIL simulator's capability can be further extended to support a broader class of applications. The collaboration among all the stakeholders in the CubeSat Community is necessary to achieve the final goals of improving mission reliability and enhancing CubeSat performance.

Chapter 4 reference

1. S. Corpino, S. Chiesa, F. Stesina, N. Viola, “*Small satellite and CubeSat missions: ten years of activities at Politecnico di Torino*”, 2nd IAA Conference on University Satellite Mission and CubeSat Workshop, Rome (Italy), 2013.
2. Toorian, K. Diaz, S. Lee, “*The cubesat approach to space access*”, Proc. IEEE Aerospace Conference, pp. 1-14, 2008.
3. CubeSat Design Specification Rev12, The CubeSat Program, Cal Poly SLO, 2009. http://www.cubesat.org/images/developers/cds_rev12.pdf, last access July 2013.
4. D. Selva, D. Krejci, *A survey and assessment of the capabilities of CubeSats for Earth observation*. 5-6, Acta Astronautica : Elsevier, 2012, Vol. 74.
5. R., Sandau, *Status and trends of small satellite missions for Earth observation*. 1-2, Acta Astronautica : Elsevier, 2010, Vol. 66.
6. *The prototype development phase of the CubeSat On-Board Processing Validation Experiment (COVE)*. Wilson, Thor. Big Sky, Montana : IEEE Aerospace Conference, 2012.
7. C. Duncan *Low Mass Radio Science Transponder – Navigation Anywhere*. Cambridge, Massachusetts : Interplanetary CubeSat Workshop, May 2012.
8. *ElaNa missions selected*. NASA. [Online] [Cited: 2012, 5-September.] http://www.nasa.gov/directorates/heo/home/CSLI_selections.html.
9. M.W. Smith, et al. *ExoplanetSat: Detecting transiting exoplanets using a low-cost CubeSat platform: SPIE Proceedings, July 2012*.
10. W. Blackwell, et al. s.l. , *Nanosatellites for earth environmental monitoring: The MicroMAS project*. Microwave Radiometry and Remote Sensing of the Environment (MicroRad), 2012, March.
11. J. Cutler et al. *The Radio Aurora Explorer – A Bistatic Radar Mission to Measure Space Weather Phenomenon Utah* : 24th Annual Conference on Small Satellites, 2010.
12. S. Palo et al., *The Colorado Student Space Weather Experiment: A CubeSat for Space Physics*. Bremen, Germany : 38th COSPAR Scientific Assembly, 2010.
13. Hawaii, University of. *Ho' Opononono- a Radar Calibration CubeSat Mission*. [Online] [Cited: 2012, 5-September.] <https://directory.eoportal.org/web/eoportal/satellite-missions/h/hooononono>.
14. P.C. Galeone, R. Walker R., V. Gupta, A. Kinnaird, J. Vannitsen, *CubeSats launched on the Vega qualification flight*. Portoroz : 4S Symposium, 2012.
15. J. Bouwmeester, J. Guo, *Survey of worldwide pico and nanosatellite missions, distributions and subsystem technology*. Bouwmeester J., Guo J. 7-8, Acta Astronautica : Elsevier, 2010, vol. 67.
16. QB50 project. [Online] www.qb50.eu.
17. Humsat project. [Online] www.humsat.org.
18. F. Aguado Agelet, J. Puig-Suari et al. *HumSat: Humanitarian Satellite constellation. A nano satellite constellation for climate change monitoring and humanitarian initiative*. Prague, CZ : 61st International Astronautical Congress, 2010.
19. G. Obiols-Rabasa, S. Corpino, F. Nichele, G. Ridolfi, F. Stesina, N. Viola *N.3-star program at Politecnico di Torino*. Portoroz : Small Satellites Systems and Services - 4S Symposium, 2012.
20. J. Bouwmeester, J. Guo, “*Survey of worldwide pico- and nanosatellite missions, distributions and subsystem technology*”, Acta Astronautica, vol. 67(7-8), pp. 854-862, 2010.
21. K. Woellert, P. Ehrenfreund, A. J. Ricco, H. Hertzfeld. “*CubeSats: Cost-effective science and technology platforms for emerging and developing nations*”, Advances in Space Research, vol. 47, pp. 663-684, 2011
22. R. Staehle, et al., “*Interplanetary CubeSats: Opening the Solar System to a Broad Community at Lower Cost*”, *Journal of Small Satellites*, vol.2(1), pp. 161-186, 2013
23. B. Klofas, J. Anderson, K. Leveque, *A survey of CubeSat communication systems*,. San Louis Obispo: CubeSat Developers Conference, 2008.
24. G. F. Dubos, J. Castet, J. H. Saleh, “*Statistical reliability analysis of satellites by mass category: does spacecraft size matter?*”, Acta Astronautica, vol. 67(1-2), pp. 584-595, 2010.

25. M. Cho, H. Masui, T. Hatamura, K. Date, S. Horii, S. Obata, “*Overview of Nano-satellite Environmental Tests Standardization Project: Test Campaign and Standard Draft*”, 26th Annual AIAA/USU Conference on Small Satellites, 2012.
26. N. K. Ure, Y. B. Kaya, G. Inalhan, “*The Development of a Software and Hardware-in-The-Loop Test System for ITU-PSAT II Nano Satellite ADCS*”, Proc. IEEE Aerospace Conference, Big Sky, MT, 2011.
27. M. Raif , U. Walter, J. Bouwmeester, “*Dynamic system simulation of small satellite projects*”, Acta Astronautica, vol.67, pp 1138–1156, 2010.
28. F. Stesina, S. Corpino, R. Mozzillo, G. Obiols Rabasa, “*Design of the active attitude determination and control system for e-st@r CubeSat*”, Proceedings of the 63rd IAC, Naples (Italy), 2012.
29. A. K. Hayder, R. L. Wiley, G. Halpert, D. J. Flood, S. Sabripour, *Spacecraft Power Technologies*, London, UK: Imperial College Press, 2003.
30. Computer Science Corporation (Members of the Technical Staff Attitude Systems Operation), *Spacecraft Attitude Determination and Control*, J. R. Wertz, Ed. Dordrecht, The Netherlands: D. Reidel Publishing Company, 1978.
31. M. Polites, C. Quarles, D. Kaderbek, “*Pulse Width Modulating Low Power Magnetic Torquers for Precise Spacecraft Attitude Stabilization*”, Proceedings of the Institution of Mechanical Engineers Part G: Journal of Aerospace Engineering, vol.219, pp. 471-482, 2005.
32. D. G. Gilmore, *Spacecraft Thermal Control Handbook*, vol. 1: Fundamental Technologies, D. G. Gilmore, Ed. Reston, VA: American Institute of Aeronautics and Astronautics, Inc., 2002.
33. International Association of Geomagnetism and Aeronomy (Members of the Working Group V-MOD), “*International Geomagnetic Reference Field: the eleventh generation*”, *Geophysical Journal International*, vol.183(3), pp 1216–1230, 2010
34. <http://www.cadsoftusa.com/>
35. <http://www.tek.com/oscilloscope/tbs1000-digital-storage-oscilloscope>
36. <http://www.elpa.it/>
37. *Space Mission Analysis and Design*, 3th Edition, J. Wertz, W.Larson, Space Technology Library, ISBN 1-881883-10-8 , 1999
38. R. Wisniewski, (1995a). *Influence of aerodynamic torque on ørsted satellite motion*. Technical Report TN-251. Aalborg University
39. J. Springmann, J. Cutler, H. Bahcivan, *Magnetic Sensor Calibration and Residual Dipole Characterization for Application to Nanosatellites*, Toronto, Ontario Canada, 2010, August 2010.
40. <http://www.samtec.com/>
41. <http://www.xsens.com/en/general/mti>
42. S.Corpino, F.Stesina, R.Mozzilo, L.Feruglio, F. Nichele, G.Obiols Rabasa, *CubeSat Design Description (CDD) of e-st@rII*, Final Acceptance Data Package for ESA Education Office, October 2013.
43. R. Mozzillo, F. Stesina, L.Feruglio, *TR_Estar2.012013_IMU characterization*, Final Acceptance Data Package for ESA Education Office, December 2012

Chapter 5. Conclusion

The research aims at the design and verification of complex systems with particular attention to the challenging system Guidance Navigation and Control (GNC) for space vehicles. The main topic is the development of a tool (the simulator StarSim) associated to a methodology that has been studied and developed to improve the design and V&V process in terms of effectiveness, cost-reduction, and reliability without loss of performance.

The proposed methodology pursues a Model and Simulation-based approach that is making inroads in System Engineering field unlike the document-centric approach.

The simulator has been developed to support the engineering team with a versatile tool which can be used throughout the design and development process of a space system. Thanks to its modular architecture, the simulator shows a good flexibility with respect to different project phases and types of missions/systems. Simulation virtual models may be uploaded in the simulator according to the specific design stage: models with different levels of detail and complexity are available and new models can be implemented and integrated in the simulator in the future. Software and hardware interfaces have been developed in order to cover the most common application cases (e.g. (un)named pipes, serial, USB, LAN). Power and telecommunication Ground Support Equipment has been designed and built (according to the limited budget), mainly to support the case study but it can easily be re-use in future programs. In fact, another advantage of the methodology is the possibility of re-using models, tools and items both from a certain phase of the product life cycle to the next ones and from past programs to future ones.

The type of mission of interest can also be chosen by setting the simulator in terms of specific mission parameters and systems involved in the simulation session.

Another key feature of the simulator is the use of freeware. Python has been used to manage the software and firmware of StarSim: the user interface, the simulation architecture construction, the databases and data repository handling. C/C++ is the language in which any virtual model is written, and, above all, the language in which the simulation code and the on board software are generated.

Well known commercial products have been only used to “validate” the HIL simulations results wherever possible.

StarSim has been used for the verification of the functional and operational requirements of the e-st@r CubeSats. A great number of simulation sessions have been performed in different “in-the-loop” configurations (AIL, SIL, CIL HIL and Hybrid among the previous). At the end of the verification campaign, the CubeSat has proven to be compliant with the required functions and to operate as expected.

A large number of requirements have been verified within a single simulation session, thus saving resources in terms of time for test set-up and running, man-hours and costs, if compared to other testing techniques. Moreover, the test has allowed following out the behaviour of the equipment included in the simulation loop. For example, the actual power consumption of each piece of hardware has been measured and used to refine the power budget. AIL, SIL and CIL simulation have concurred to design and verify step-by-step the determination and control algorithms, software and, finally, the hardware (microcontroller, sensors, actuators and other items and devices).

The HIL testing results have been compared with AIL and SIL simulations to “validate” the methodology and the facility. These comparisons have shown a good correlation between the different simulations, even if some differences have been noticed. For example, using the HIL facility has helped to identify unexpected interactions between different devices during the integration of the CubeSat. The design has been reviewed or some requirements have been negotiated to eliminate these potential failure causes, thanks to the information provided by the HIL simulations.

The first release of the simulator (with the related specifications and the first validation through a case study) is the final target of this thesis that has been successfully met. In fact, StarSim 1.0 has been completed from the design point of view and the simulator software version 1.x can be delivered.

The present thesis lays the groundwork for future works. At the moment, simple designs can be investigated through this platform, but the platform can be extended for testing more complex missions without re-designing the simulator's architecture.

Some improvements can enhance the simulator performance and accuracy; general suggestions may be provided according to the PhD activities experience, the state of the art and the ESA indications:

1. to make the highest number of functions within the methodology process automatic: i.e. the capability to acquire data directly from the design outputs and the verification of requirements;
2. to update databases: virtual models and interfaces are now sufficiently rich to perform an entire simulation campaign for a CubeSat and they are already outfitted for the feasibility and first design steps simulation of RV spacecraft and launch vehicles; most common interfaces are already implemented in the simulator, but other protocols could be easily included, e.g. the integration of the CAN bus interface;
3. to improve the ground support equipment, in order to reproduce the attitude dynamics and relative linear motion between two objects: i.e. the addition of a robotic arm will improve attitude simulation capability;
4. to upgrade the simulator for the design and verification of the GNC system for other missions, spacecraft and onboard subsystems: in fact, StarSim may be tailored for different applications without major changes and, in principle, it can handle simulation of several kinds of space missions and systems.

In conclusion, it has been proven that the methodology is effective and a modular and flexible simulator (StarSim v1.x) can support the design and verification process for GNC systems of space vehicle through the application of the methodology. Moreover it has been demonstrated that improving the design verification activity is a key factor for the development of innovative missions, as the CubeSat missions, and leads to higher confidence in the success of a space mission.

Appendix A: Reference Frames

The GNC needs the definition of unambiguous reference frames attitude measurements, guidance and control as well as for orbit navigation. Reference frames can be inertial or not inertial and they can be associated to sensor s or actuators for attitude measurement and control or they refer to a guidance target. For a detailed discussion on reference frames, the ECSS-E-ST-10-09 is the reference taken into account in the development of this thesis.

It is possible to identify different types of coordinate systems based on the number and type of the coordinates. The most common three-dimensional coordinate systems are:

Cartesian: also called rectangular, it is formed by three perpendicular lines whose intersection identifies the origin. Each of the three lines, normally indicated as X , Y and Z , has an associated unit and direction. The generic coordinates of a point in the space are indicated with the letters x , y and z and the three coordinates are written with the symbol (x,y,z) .

Cylindrical: in this reference system the coordinates are ρ , ϕ and z . Considering a generic point P , and its projection Q on the X - Y plane, the coordinate z indicates the distance PQ is the distance from the origin and the point Q , while ϕ is the angle between the vector ρ and the X axis.

Spherical: it is based on the coordinates ρ , θ and ϕ . Considering a generic point P , and its projection Q on the X - Y plane, the coordinate ρ indicates the distance of P from the origin, θ is the angle between the vector ρ and the Z axis and, calling ρ' the vector that connect the origin and the point Q , ϕ is the angle that this vector form with the X axis.

It is possible to describe the same point in more than on coordinate system, so a set of transformations exist in order to be able to change from a coordinate system to another. Now the different frames used in this report are defined.

Inertial Reference Frames

An Inertial reference frame is a coordinate system in which is verified Newton's first law: with an acceptable approximation the so called *fixed star reference frame* is considered inertial and it includes the Sun, the stars and every other body with a uniform rectilinear motion as regards to it (not accelerating or rotating).

Earth-Centered Inertial (ECI) frame

Considering the Earth as third body, the reference frame obtained cannot be considered as a real Inertial Reference Frame because of Earth's revolution and rotation movements. In particular, the rotary motion submits the objects on the surface of the Earth far from the poles to a little centrifugal force. However this acceleration can be neglected in some cases and the Earth considered, with a good approximation, as an inertial reference frame (called ECI). The ECI frame is an inertial frame used for navigation. It is fixed in space and its origin is located at the center of the Earth with the Z -axis pointing towards the North Pole. The X -axis points towards vernal equinox, the point where the plane of the Earth's orbit around the Sun crosses the Equator going from south to north, and the Y -axis completes the right hand Cartesian coordinate system. All the different motions of the satellite could be presented in this frame, but only the velocity of the Orbit frame and the motion of the Sun are directly compared to this frame. The frame is denoted I .

The rotary motion, present in reality, brings objects far from the equator to the so called Coriolis force that causes a deviation of the motion of every object towards right in the north hemisphere and towards left in the south hemisphere, as demonstrated by the well known Foucault pendulum. For every object orbiting the Earth it is possible to define an inertial reference frame based on the ECEF reference frame with the following rotation matrix:

$$R_I^E = \begin{bmatrix} \cos \alpha & \sin(\alpha) \cos(\beta) & \sin(\alpha) \sin(\beta) \\ -\sin(\alpha) & \cos(\alpha) \cos(\beta) & \cos(\alpha) \sin(\beta) \\ 0 & -\sin(\beta) & \cos(\beta) \end{bmatrix}$$

Where α is the angle given by $\omega_e t$, with ω_e the Earth rotational speed and t the time and β is the angle given by the following equation:

$$\beta = (23.439281083 - \frac{46.815}{6300} \text{ Juliancenturies}) \frac{\pi}{180}$$

In which Juliancenturies is:

$$\text{Juliancenturies} = \frac{\text{day2000} - 2451545}{36525}$$

$$\text{day2000} = 2451543.5 + (\text{year} - 2000)365 + 1 + \text{floor}\left(\frac{\text{year} - 2000}{4}\right) + \text{day}$$

Non-Inertial Reference Frames

A Non-Inertial reference frame is a coordinate system in which the description of the dynamic of objects does not verify the principle of inertia. It is a system in which an object subject to a resultant of forces equal to zero however has a non-uniform motion. All and only the reference frames that move of accelerated motion in reference to the fixed star reference frame have this property and can be defined as Non-Inertial.

Earth-Centered Earth-Fixed (ECEF) frame

The ECEF reference frame has its origin located in the center of the Earth but the X and Y axes rotate with the Earth relative to the ECI frame. This rotation is around the Z-axis, both of the ECI and the ECEF frame, and has a rate of

$$\omega_e = \frac{1 + 365.25 \text{cycles}}{(365.25)(24)h} \frac{2\pi \text{rad} / \text{cycle}}{3600s / h} \approx 7.292115 * 10^{-5} \text{ rad} / s$$

The Z-axis points towards the North Pole, X-axis points toward the intersection between the Greenwich meridian and the Equator, which is at 0° longitude and 0° latitude, and the Y-axis completes the right handed orthogonal system. The frame is denoted E.

North-East-Down (NED) frame

The North-East-Down reference frame is one of the Geodetic coordinate systems. It is a local reference frame and it depends on the position on the Earth. The X-Y plane coincides with the local horizon and it has unit vectors pointing the local North and East; the Z axis completes the right-handed triad pointing toward the center of the Earth (Down). Thanks to this property the NED reference frame is one of the most utilized for Earth surface studies. The frame is denoted N.

It is possible to define a rotation matrix in order to pass from the ECEF to the NED reference frame as:

$$R_E^N = \begin{bmatrix} -\sin(\text{lat}) \cos(\text{lon}) & -\sin(\text{lat}) \sin(\text{lon}) & \cos(\text{lat}) \\ -\sin(\text{lon}) & \cos(\text{lon}) & 0 \\ -\cos(\text{lat}) \cos(\text{lon}) & -\cos(\text{lat}) \sin(\text{lon}) & -\sin(\text{lat}) \end{bmatrix}$$

Orbital frame

The origin of this frame coincides with the center of mass of the satellite. It rotates relative to the ECI frame, with a rate of ω_o depending on the altitude of the orbit.

The X-Z plane is the orbital plane with unit vectors pointing one in the direction of the orbital velocity of the satellite and the other as the local vertical, while the Y-axis is orthogonal to this plane and complete the right-handed triad. The frame is denoted O.

It is possible to convert the coordinates of a point from the Orbital to the Inertial reference frame with the following rotation matrix:

$$R_o^I = \begin{bmatrix} c(\Omega)c(\omega) - s(\Omega)s(\omega)c(i) & -c(\Omega)s(\omega) - s(\Omega)c(\omega)c(i) & s(\Omega)s(i) \\ s(\Omega)c(\omega) - c(\Omega)s(\omega)c(i) & -s(\Omega)s(\omega) - c(\Omega)c(\omega)c(i) & -c(\Omega)s(i) \\ s(\omega)s(i) & c(\omega)s(i) & c(i) \end{bmatrix}$$

with c and s compact notations for \cos and \sin . The other variables here used are defined in the Appendix B, in which is treated the description of the orbit.

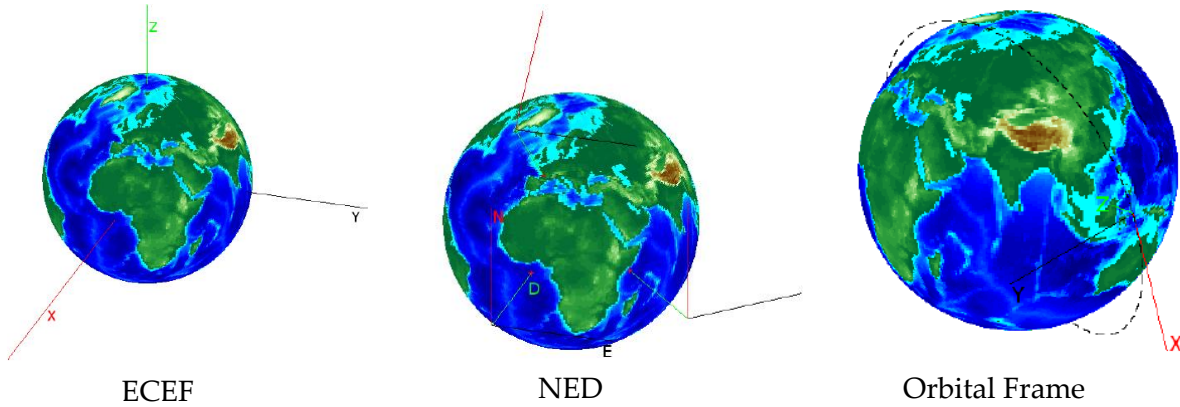


Figure 181: Reference frames

Body frame

The body frame is fixed to the satellite and for practical reasons the origin is placed at the satellite's center of mass. The axes are locked in the satellite, X-axis is forward, Z-axis is downwards and the Y-axis completes the right-hand orthogonal system. This frame is denoted B.

In order to transform a set of coordinates from the Body to the Orbital reference frame, or vice versa, the introduction of a rotation system is needed. The classical rotations used to describe these transformations are based on the use of the quaternion or the Euler Angles, both described in Appendix C.

Sensor frame

These coordinate frames are used to describe translations and rotations of the spacecraft w.r.t. location and direction of equipment, such as sensors, thrusters or mechanism.

The origin is in a particular point on the spacecraft, e.g. the point defining the origin of the spacecraft coordinate system, the centre of the docking port, defining the docking frame, the centre of a sensor, defining the measurement frame.

The axes are coordinate-aligned with, or under a fixed angle to, the body frame.

The transformation from the spacecraft attitude frame to one of the geometric frames is a parallel shift in x, y, z from the centre of mass of the spacecraft and a fixed rotation $\varphi_x, \varphi_y, \varphi_z$ around the origin of the frame. It requires, therefore, the knowledge of the instantaneous position of the CoM of the spacecraft.

Appendix B: Attitude representation

There are some different ways to represent the attitude of the satellite in a reference frame. These, along with tools to convert between the frames, are described here.

Euler Angles

The Euler Angles can be used to describe the angular position of a Body reference frame XYZ , with a set of rotations, relative to another reference frame xyz considered fixed. Here only rotations are considered, so the two reference frame are taken so that the origin is the same for both of them. If the x - y and X - Y planes do not coincide, a line of intersection will exist and it is called line of nodes (N). If they coincide then the line of nodes is taken coincident with the X axis. The Euler Angles are:

- α is the angle between the line of nodes and the x -axis, it is called *precession*;
- β is the angle between the z -axis and the Z -axis, it is called *nutation*;
- γ is the angle between the line of nodes and the X -axis, it is the intrinsic *rotation*.

The Euler Angles allow a representation of the rotation matrix in a easy form obtained with a multiplication of three rotation matrices. In other words the complete rotation described above can be done in three distinct passages:

- rotation around the z -axis of an angle α , to obtain the x -axis coinciding with the line of nodes N :

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

- rotation around the line of nodes N of an angle β :

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

- rotation around the Z -axis of an angle γ :

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

The sequence described above is only one of the twelve possible sequences describing same rotation.

$$\begin{bmatrix} \hat{X} \\ \hat{Y} \\ \hat{Z} \end{bmatrix} = R_{\gamma} R_{\beta} R_{\alpha} \begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{bmatrix}$$

$$\begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{bmatrix} = R_{\alpha}^T R_{\beta}^T R_{\gamma}^T \begin{bmatrix} \hat{X} \\ \hat{Y} \\ \hat{Z} \end{bmatrix}$$

It is called ZXZ from the axes around which the rotations take place. The other possibilities are XZX , YXY , YZY , ZYZ , XZY , XYZ , YXZ , YZX , ZYX and ZXY . These sequences are obtained from all the possible permutations of not consecutive equal axes.

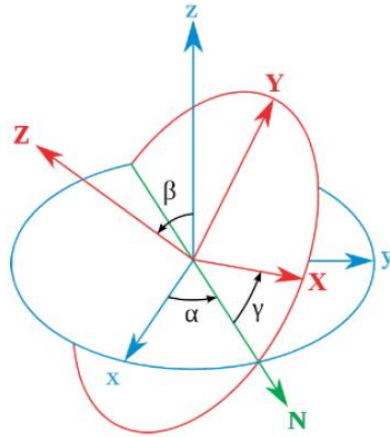


Figure 182: Euler angles representation

Quaternion

The main reason for using unit quaternion instead of Euler parameters is to avoid singularity which can occur when using Euler angles. In math quaternion are a numerical system used in order to extend the complex numbers. They were introduced for the first time by Sir William Rowan Hamilton in 1843 and successively applied to mechanics in three-dimensional spaces. One of the principal properties of quaternion is the fact that the product of two quaternion is not cumulative, which means that the product depends on the order of its terms. Hamilton defined a quaternion as the quotient of two vectors, but they can be also represented as sum of a scalar number and a vector. Quaternion are used vastly by theoretical and applied math, specially for rotations in the three-dimensional space. A generic quaternion can be written as:

$$\vec{q} = q_0 + q_1\hat{i} + q_2\hat{j} + q_3\hat{k} = a + b\hat{i} + c\hat{j} + d\hat{k}$$

with a, b, c and d real numbers.

Quaternion contains naturally real numbers if $b = c = d = 0$ ($q = a$) and complex numbers if $c = d = 0$ ($q = a + b\hat{i}$).

From the Euler theorem, that guarantees the possibility to rotate a fixed reference frame on another arbitrary reference frame with a simple rotation around an axis a (also called Euler rotation axis) fixed in both reference frames, it is possible to adopt quaternion to define any change of reference system in the three-dimensional space.

Thanks to their properties it is possible to represent uniquely any rotation without having degenerating points, that are points in which at least a parameter lose its meaning. The rotation of an angle α around a generic axis u can be described introducing the Euler parameters and obtaining the quaternion:

$$\vec{q} = \cos\left(\frac{\alpha}{2}\right) + \vec{u} \sin\left(\frac{\alpha}{2}\right)$$

or in matrix form:

$$\bar{q} = \begin{bmatrix} \bar{q} \\ q_4 \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{\alpha}{2}\right) \\ \hat{u}_1 \sin\left(\frac{\alpha}{2}\right) \\ \hat{u}_2 \sin\left(\frac{\alpha}{2}\right) \\ \hat{u}_3 \sin\left(\frac{\alpha}{2}\right) \end{bmatrix}$$

Using quaternion above introduced, it is possible to define rotation matrix as:

$$R_{q^0:\bar{q}} = I_{3 \times 3} + 2q_4 S(\bar{q}) + 2S^2(\bar{q})$$

where $S(q)$ is the skew-symmetric matrix, that is:

$$S(\bar{q}) = \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix}$$

The last two equations allow to write rotation matrix from body to orbital frame as:

$$R_B^O = I_{3 \times 3} + 2q_4 S(\bar{q}) + 2S^2(\bar{q})$$

so a representation of the rotation matrix from Orbital to Body frame can be calculated as follow:

$$R_O^B = (R_B^O)^T = \begin{bmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1 q_2 + q_3 q_4) & 2(q_1 q_3 - q_2 q_4) \\ 2(q_1 q_2 - q_3 q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2 q_3 + q_1 q_4) \\ 2(q_1 q_3 + q_2 q_4) & 2(q_2 q_3 - q_1 q_4) & -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{bmatrix}$$

This rotation matrix can also be written as:

$$R_O^B = [c_1^B \quad c_2^B \quad c_3^B]$$

$c_i^B = [c_{ix}^B, c_{iy}^B, c_{iz}^B]$ are column vectors containing directional cosines.

Appendix C: In orbit disturbances that affect the GNC operations

The space environment strongly affects the GNC project and imposes constraints on what type of control methods are more effective. An example is the EMF that is a great source of disturbance for in LEO objects but passive permanent magnets or magnetic actuators take advantage from EMF to control attitude. Another example is the Earth's gravity field that attracts on objects in the Earth center: it causes the deorbiting and, in general, disadvantages the attitude control of flattened spacecraft but it results very useful when Earth pointing is a mission requirement.

In literature, GNC sections list, explain, and model the space phenomena that affect the spacecraft dynamics and kinematics both linear and rotational: the main four source of disturbance are:

- EMF: it is generated by the core of the Earth that interacts with the magnetic moment caused by electronic devices on board. In fact, a electronics circuit can be approximated as a coil in which flowing current generating a dipole moment perpendicular to the plane of the circuit.
- Atmospheric drag
- Gravity: in the essential uniform gravity, the difference between center of mass and center of gravity is indistinguishable, but in free fall of a space orbit gravity gradient torque are caused when a spacecraft center of gravity is not aligned with the center of mass wrt the local vertical. The gravity gradient torque increases with the angle between the local vertical and the spacecraft's principal axis, always trying to align the minimum principal axis with the local vertical.
- Solar radiation pressure

Other kinds of disturbance should be considered even if they do not directly hit the dynamics aspects. This is the case of the radiations that can interact with electronics devices and sensors provoking them transient or permanent misbehaviors. E.g. CCDs camera have bad behavior within the Van Allen belts and APSs (Active Pixel Sensor) are more robust; the Sun radiation is very useful for sun sensor measurements but, like the Moon, it avoids the star sensors measurements. In general, thermo vacuum environment can degrade the electronic devices performance: outgassing phenomenon can waste lens of earth or star sensors; Occasionally, what is normally neglected becomes strangely large, for this reason the GNC design shall be sufficiently robust to reject the undesirable behaviors due to disturbances.

Sometimes literature forgets internal disturbances, where *internal* means caused by the internal phenomena. It is possible to categorize them from their causes:

- Fuel sloshing that change the spacecraft mass properties, in particular the inertia
- Rotating mechanisms that cause undesired torques
- Electronics devices that provoke magnetic dipole generating, together the EMF, a residual magnetic torque
- Dynamics of flexible body and deployment of antennas and solar arrays that changes the spacecraft mass properties and causes forces and torques
- Thruster misalignment and mismatch of thruster outputs

Internal disturbances often become comparable with the external ones for intensity and influences the control design. For this reason, they should be modelled using verified parameters (e.g. manufacturer data) or parameters identified by dedicated tests. Worst case parameters can be used to avoid tests provided that robustness is demonstrated w.r.t these parameters.

Appendix D: Control techniques

PID (Proportional Integrative Derivative) Control

PID: is the most simple and widely used controller Table 39 lists the different combination of the proportional, integrative and derivative actions. Note that $u(t)$ is the control variable and $e(t)$ the error signal.

	Continuous	Discrete
P	$u(t) = K_p e(t)$	$u(k) = K_p e(k)$
I	$u(t) = \frac{K_p}{T_i} \int_{t_0}^t e(t) dt$	$u(k) = u(k-1) + \frac{K_p T}{T_i} e(k)$
D	$u(t) = K_p T_d \frac{de(t)}{dt}$	$u(k) = u(k-1) + \frac{K_p T_d}{T} (e(k) - e(k-1))$
PID	$u(t) = K_p \left[e(t) + \frac{1}{T_i} \int_{t_0}^t e(t) dt + T_d \frac{de(t)}{dt} \right]$	$u(k) = K_p \left[e(k) + \frac{T}{T_i} \sum_{i=0}^{k-1} e(i) + \frac{T_d}{T} [e(k) - e(k-1)] \right]$

Table 39: PID

Optimal Control

Optimal control deals with the solution of one of the most celebrated problems of modern control theory. This problem is defined as the determination of the best possible control strategy (usually of the optimum control vector $u(t)$), which minimizes a certain cost function or performance index. Starting from the linear systems state space equations, the objective of the optimal control problem is to determine a control vector $u(t)$ which will “force” the behavior of the system under control to minimize some type of cost function, while at the same time satisfying the physical constraints of the system.

Cost functions can be for :

- the **minimum time control**: $J = \int_{t_0}^t dt = t - t_0$
- the **terminal control**: $J = [x(t_f) - x_d(t_f)]^T S [x(t_f) - x_d(t_f)]$, where x_d is the final desired value and S is a real, symmetric, positive semi-definite weighting matrix.
- The **minimum control effort**: $J = \int_{t_0}^t u^T(t) R(t) u(t) dt$, where $R(t)$ is a symmetric, positive, definite weighting matrix for $t \in (t_0, t_f)$
- The **optimal control servomechanism or tracking**:
 $J = \int_{t_0}^{t_f} [x(t_f) - x_d(t_f)]^T Q(t) [x(t_f) - x_d(t_f)] dt = \int_{t_0}^{t_f} e^T(t) Q(t) e(t) dt$, where $Q(t)$ is a real symmetric, positive semi-definite, weighting matrix.
- The **optimal regulator**: $J = x^T(t_f) S x(t_f) + \int_{t_0}^{t_f} [x(t)]^T Q(t) [x(t)] - [u(t)]^T R(t) [u(t)] dt$, where $S, Q(t), R(t)$ are weighting matrices.

LQR

From a mathematical point of view. The optimal regulator problem may be formulated as follows. Consider the linear, time-varying system described in state space by the vector differential equation:

$$\dot{x}(t) = A(t)x(t) + B(t)u(t) \text{ and } x(t_0) = x_0$$

Find a control signal $u(t)$ which minimizes the cost function:

$$J = \frac{1}{2} x^T(t_f) S x(t_f) + \frac{1}{2} \int_{t_0}^{t_f} [x(t)]^T Q(t) [x(t)] - [u(t)]^T R(t) [u(t)] dt$$

This criterion is a sum of inner products of the vectors $x(t)$ and $u(t)$, and for this reason it is called the quadratic cost function. The matrices S , $Q(t)$, and $R(t)$ are weighting matrices and are chosen to be symmetric. Here, we stress again that the main reason for including the energy-like quadratic terms $[x(t)]^T Q(t) [x(t)]$ and $[u(t)]^T R(t) [u(t)]$ in the cost function J is to minimize the dissipated energy in the system and the required input energy (control effort), respectively. The quadratic term $x^T(t_f) S x(t_f)$ is included in J to force the final value $x(t_f)$ of $x(t)$ to be as close as possible to the equilibrium point of the system. Note that $x(t_f)$ is unspecified. The minimization of the cost function J will be done using the method of maximum principle. Defining the Hamiltonian:

$$H(x, u, \lambda, t) = \frac{1}{2} [x(t)]^T Q(t) [x(t)] + \frac{1}{2} [u(t)]^T R(t) [u(t)] + \lambda^T(t) [A(t)x(t) + B(t)u(t)]$$

where $\lambda(t)$ is the vector of Lagrange multipliers. Now, it is possible to define a cost function J' :

$$J' = \frac{1}{2} x^T(t_f) S x(t_f) + \int_{t_0}^{t_f} [H(x, u, \lambda, t) - \lambda^T(t) \dot{x}(t)] dt. \text{ It is proven that a necessary condition for } J \text{ to}$$

maximum or minimum is that the derivate of J' is equal to 0. Consequently, x and u vectors have to satisfy the equation $\delta J' = 0$, in which the following relations should hold:

$$\begin{aligned} \frac{\partial H}{\partial x} &= -\dot{\lambda}(t) = Q(t)x(t) + A^T(t)\lambda(t) \\ \frac{\partial H}{\partial u} &= 0 = R(t)u(t) + B^T(t)\lambda(t) \\ \frac{\partial H}{\partial \lambda} &= A(t)x(t) + B(t)u(t) \\ \left. \frac{\partial \theta}{\partial x} \right|_{t=t_f} &= Sx(t_f) = \lambda(t_f) \end{aligned}$$

The first three conditions represents the boundary conditions of the problem, that solved with respect to $u(t)$ carries out to:

$$u(t) = -R^{-1}(t)B^T(t)\lambda(t)$$

The last equation can be expressed as a linear state feedback law, resulting: $u(t) = K(t)\lambda(t)$. Moreover, assuming that the vector of Lagrange multipliers (called *costate*) is linear and, through skipped mathematical operations, the Riccati equation is obtained:

$$\dot{P}(t) + P(t)A(t) + A^T(t)P(t) - P(t)B(t)R^{-1}(t)B^T(t)P(t) = -Q(t)$$

where $Q(t)$ is a symmetric matrix and P is an unknown matrix. Solving this equation the $K(t)$ is determined $K(t) = -R^{-1}(t)B^T(t)P(t)$.

Figure 183 shows the block diagram of the linear optimal regulator

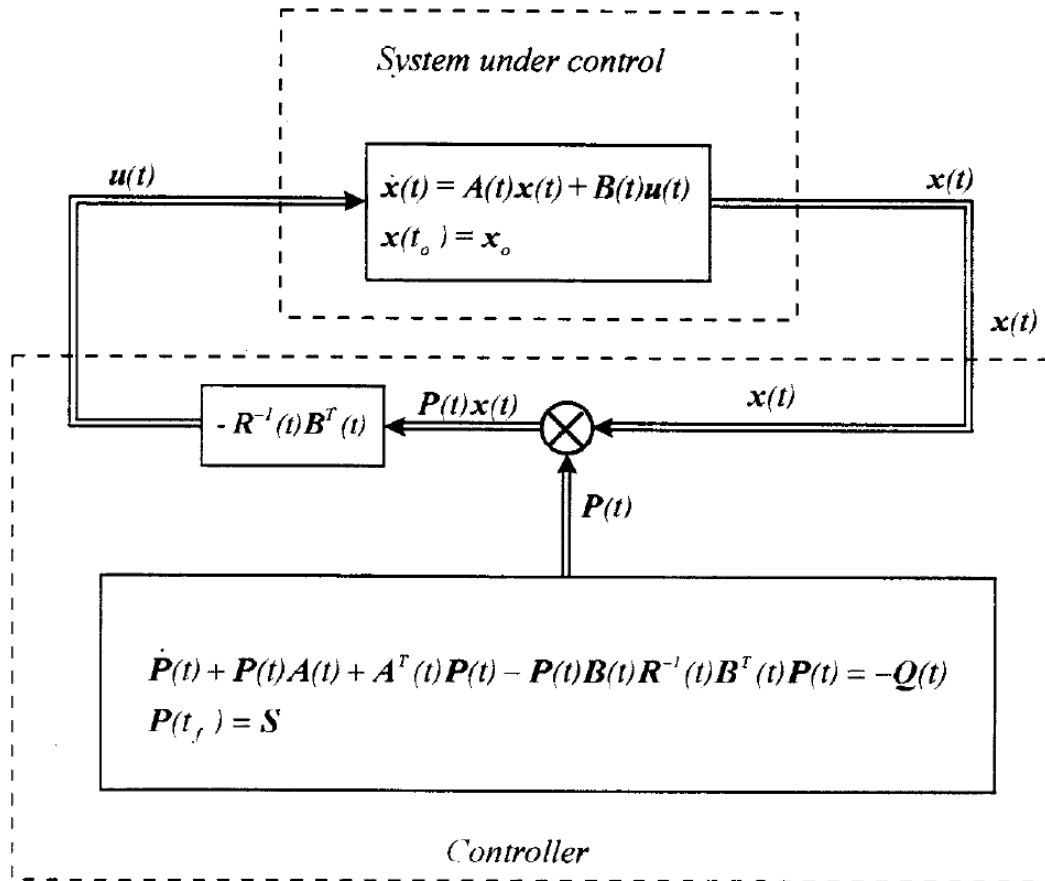


Figure 183: Block diagram of the optimal linear regulator, using Riccati equation

Hinfinity control

The Hinfinity control is the most common robust control among a wide number of possibilities. The robust control problem is to find a control law which maintains system response and error signals within prescribed performances despite the effects of the uncertainty on the system. Forms of uncertainties, as already presented in the Chapter 2 of this thesis, include:

- Measurements noise
- Disturbance effects on the plant
- Modeling error due to non-linearity
- Modeling error due to time-varying parameters

It has been demonstrated that non linear functions can be linearized for small perturbation about an operating point. It is therefore possible to describe a non linear system by a series of linear models each constructed about a known operating point. If the operating point can be linked to a measurement, then a simple robust system may be constructed using a LQG approach. The feedback and Kalman gain matrices are calculated in advance for each operating point and some form of interpolation used to provide a "Gain Scheduling Controller".

In general, however, robust control system design uses an idealized or nominal model of the plant. Uncertainty in the nominal model is taken into account by considering a family of models that include all possible variations. As seen, a system is robustly stable if the controller can be found that will stabilize all plants within the family that are on the verge of instability. A controller is said to have robust performance if all the plants within the family meet a given performance specification.

With Hinf-optimal control the inputs are assumed to belong to a set of norm-bounded functions with weight. Each input in the set will result in a corresponding error. The Hinf-optimal controller is designed to minimize the worst error that can arise from any input in the set, and can be expressed as

$$\min_{\epsilon} \|e(t)\|_{\infty} = \min_{\epsilon} \sup_{\omega} |S(s)W(s)|$$

where sup is a short for supremum, which means the final result is the least upper bound. Thus the Hinfintiy optimal controller minimizes the maximum magnitude of the weighted sensitivity function over the frequency range ω or, in the mathematical terms, minimizes the ∞ -norm of the weighted sensitivity function.

Adaptive Control (main concepts)

An adaptive control system is a system which adjusts automatically on-line the parameters of its controller, so as to maintain a satisfactory level of performance when the parameters of the system under control are unknown and/or time varying.

Generally speaking, the performance of a system is affected either by external perturbations or by parameter variations. Closed-loop systems involving feedback (top portion of Figure 184), are used to cope with external perturbations. In this case, the measured value of the output $y(kT)$ is compared with the desired value of the reference signal $r(kT)$. The difference $e(kT)$ between the two signals is applied to the controller, which in turn provides the appropriate control action $u(kT)$ to the plant or system under control. A somewhat similar approach can be used when parametric uncertainties (unknown parameters) appear in the system model of Figure 184. In this case the controller involves adjustable parameters. A performance index is defined, reflecting the actual performance of the system. This index is then measured and compared with a desired performance index and the error between the two performance indices activates the controller adaptation mechanism. This mechanism is suitably designed so as to adjust the parameters of the controller (or modify the input signals in a more general case), so that the error between the two performance indices lies within acceptable bounds.

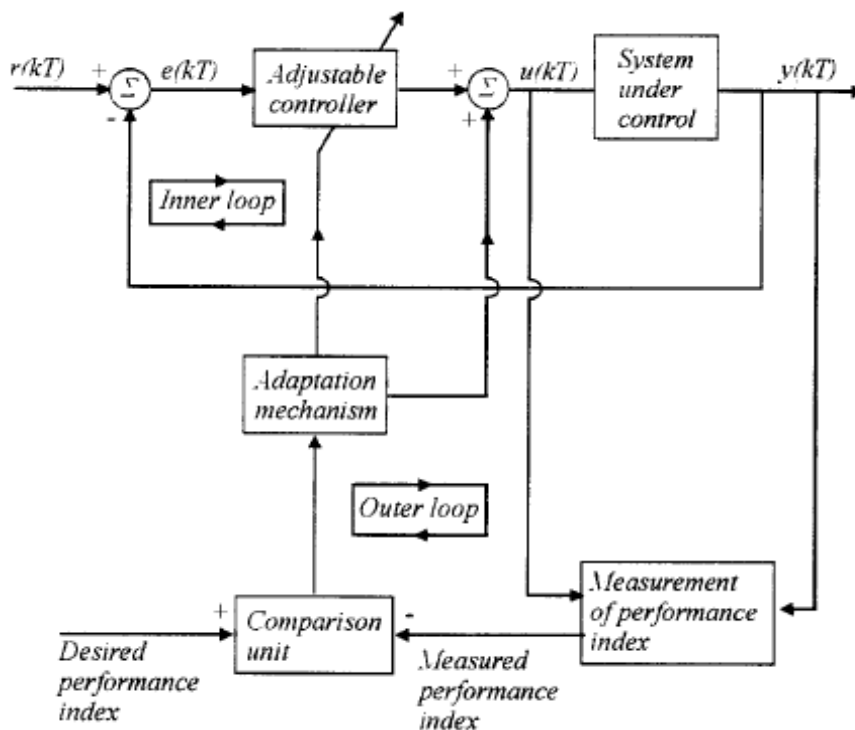


Figure 184: Adaptive Control Structure

Closer examination of Figure 184 reveals that two closed loops are involved: the “inner” feedback closed loop, whose controller involves adjustable parameters (upper portion of the figure); the supplementary “outer” feedback closed loop (or adaptation loop), which involves the performance

indices and the adaptation mechanism (lower portion of the figure). The role of the adaptation loop is to find appropriate estimates for the adjustable controller parameters at each sampling instant.

It should be mentioned that a general definition, on the basis of which one could characterize a system as being adaptive or not, is still missing. However, it is clear that constant feedback systems are not adaptive systems. The existence of a feedback loop involving the performance index of the closed-loop system is a safe rule for characterizing a system as adaptive or not.

An adaptive control system is inherently nonlinear, since the controller parameters are nonlinear functions of the measured signals through the adaptation mechanism. This is true even for the control of linear systems with unknown parameters, a fact which makes the analysis of adaptive systems very difficult. This analysis involves the stability characteristics of the closed-loop system, the satisfaction of the performance requirements, and the convergence of the parameter estimates. Adaptive control has been under investigation for many years. Major breakthroughs in the area have been reported in the last two decades. Adaptive control schemes have been applied in the paper industries, rolling mills, power plants, motor drives, chemical reactors, cement mills, autopilots for aircrafts, missiles and ships, etc. Microprocessor advances have made it quite easy to implement adaptive controllers and at low cost. The use of adaptive controllers may lead to improvement of product quality, increase in production rates, fault detection, and energy saving.

The two basic techniques to control discrete-time systems with unknown parameters are the model reference adaptive control (MRAC) scheme and self-tuning regulators (STRs).

In MRAC, a reference model is used explicitly in the control scheme and sets the desired performance. Then, an appropriate on-line adaptation mechanism is designed to adjust the controller parameters at each step, so that the output of the system converges to the output of the reference model asymptotically, while simultaneously the stability of the closed-loop system is secured. In STRs, the control design and the adaptation procedure are separate. Different parameter estimators can be combined with appropriate control schemes to yield a variety of STRs.

Model reference adaptive controllers can be either direct or indirect. The essential difference between them is that in direct MRAC the controller parameters are directly adjusted by the adaptation mechanism, while in indirect MRAC the adjustment of the controller parameters is made in two steps. In the first step, the control law is reparametrized so that the plant parameters appear explicitly in the control law. A relation between the controller parameters and the plant parameters is thus established. The plant parameters are adjusted by the adaptation mechanism. In the second step, the controller parameters are calculated from the estimates of the plant parameters.

STRs can be either explicit or implicit. In explicit STRs an estimate of the explicit plant-model parameters is obtained. The explicit plant model is the actual plant model. In implicit STRs the parameters of an implicit model are estimated. The implicit model is a reparametrization of the explicit plant model. The parameters of the implicit model and those of the controller are the same; therefore, we call the plant parameters explicit or indirect and the controller parameters implicit or direct.

Though of different nature and origin, a close relation between MRAC systems and STRS has been established. It is clear that explicit self-tuners correspond to indirect MRAC schemes, while implicit self-tuners correspond to direct MRAC schemes.

Another approach to discrete-time MRAC is that of using Lyapunov functions to prove asymptotic stability and the satisfaction of performance requirements. An expression for the error between the output of the reference model and that of the plant is formed and then the adaptation mechanism is chosen in order to make the increments of a Lyapunov candidate function negative.

The difficulty of finding an appropriate Lyapunov candidate function in the general discrete-time case restricts the use of this method. The hyperstability approach is preferable for discrete-time MRAC systems, while for continuous-time systems the Lyapunov design has mainly been used.

A first approach to MRAC was based on the gradient method. The parameter adaptation scheme obtained for synthesizing the adaptive loop was heuristically developed, initially for continuous-time systems and is known as the MIT rule.

Neural network control

The human brain is comprised of many millions of interconnected units called neurons. Each neuron consists of a cell to which is attached several dendrites (inputs) and single axon (output). The axon connects to many other neuron via connection points called synapses. A synapses produces a chemical reaction in response to an input. The biological neuron fires if the sum of the synaptic reactions is sufficiently large. The brain is a complex network of sensory and motor neurons that provide a human being with the capacity to remember thinks, learn and reason.

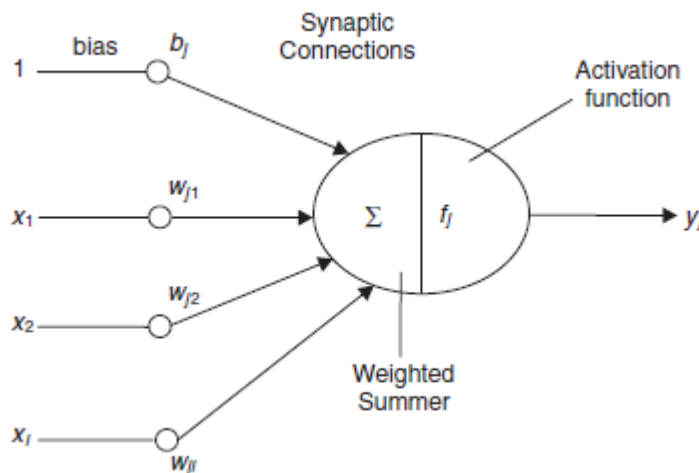


Figure 185: Basic model of a single neuron

An Artificial Neural Network try to emulate that. Along the years a great number of studies were dedicated to neural network. Artificial NN have the following potential advantages for intelligent control:

- They learn from experience rather than by programming
- They have the ability to generalize from given training data to unseen data
- They are fast and can be implemented in real time

The basic model of the single artificial neuron consists of a weighted summer and an activation function. Figure 185 shows a neuron in the j-th layer where:

$X_1 \dots X_n$ are inputs

$W_{j1} \dots W_{jN}$ are weights

b_j is a bias

f_j is the activation function

y_j is the output

The weighted sum s_j is therefore:

$$s_j(t) = \sum_{i=1}^N w_{ji} x_i(t) + b_j = W_j x + b_j$$

The activation function $f(s)$ can take many forms, some of which in the Figure 186.

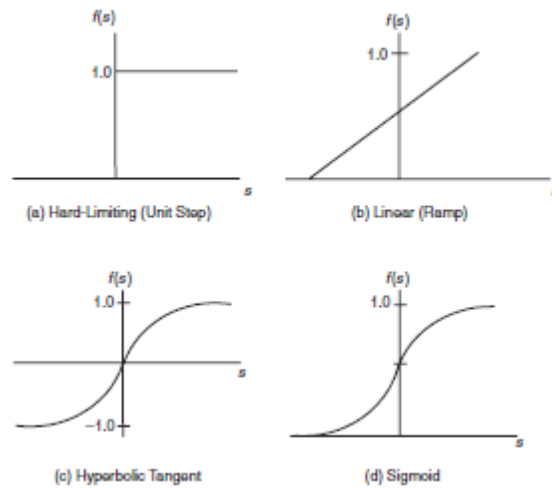


Figure 186: activation functions

From Figure 186 it can be seen that the bias b_j in the previous equation will move the curve along the s -axis i.e. effectively settings the threshold at which the neuron fires.

So in the case of hard-limiting function, if $b_j=0$, the neuron will fire when $s_j(t)$ changes from negative to positive.

The sigmoid activation function is popular for neural network applications since it is differentiable and monostatic, both of which are a requirement for the back-propagation algorithm.

Different architecture of NN has been invented and studied:

- Multilayer Perceptrons (MLP) NN
- Radial Basis Function (RBF) NN
- Fuzzy Basis Function (FBF) NN
- Sigma-Pi (SP) NN
- Supervised Growing (SG) NN
- Recurrent NN

A fundamental importance is covered by the learning in NN: learning in the context of a neural network is the process of adjusting the weights and biases in such a manner that for given inputs, the correct responses, or outputs are achieved. Learning algorithms can be divided in:

Supervised learning: the network is presented with training data that represents the range in input possibilities, together with the associated desired outputs. The weights are adjusted until the error between the actual and desired outputs meets some given minimum value.

Unsupervised learning. Also called open-loop adaptation because the technique does not use feedback information to update the network's parameters. Applications for unsupervised learning include the KSOM (Kohonen Self-Organizing Map) which is a competitive network, and Grossberg Adaptive Resonance Theory (ART) which can be used for on-line learning.

Appendix E: RD129- ARM9 processor

The CPU board adopted for 3STAR is the RD129 - ARM9 Embedded CPU (see Figure 187) produced by ELPA: it is an ARM9 CPU, designed to give good performance at a very low price for this kind of item; the same company produces also a development board (RD126, Figure 188: RD126 development board for RD129) for this CPU that gives the possibility to make all operations more convenient, providing many inputs for various devices. Features of this CPU are:

Technical features:

- 32-bits ARM 9 CPU, clocked at 240MHz;
- 32MB of SDRAM (64MB optional) at 120MHz (32-bits interfaced);
- 64MB of on board Flash memory;
- 3 Serial interfaces;
- 2 USB host interfaces (1 switchable to USB device);
- 10-bits ADC converter;
- IIC bus interface;
- 4 internal PWM timers;
- Watchdog timer;

Dimensions: very small sizes, 45x40x8 mm;

Power: 0.5W typical, 1W max; only a single 3.3Vdc 5% is required;

Operating temperature range: from -25°C to +85°C;

Development board features:

- 3.3V switching power supply;
- level translator for 2 RS232 serial interfaces;
- USB to Ethernet converter;
- USB Host connector;
- MMC or SD flash memory card connector;
- IIC bus EEPROM;



Figure 187: RD129 microprocessor

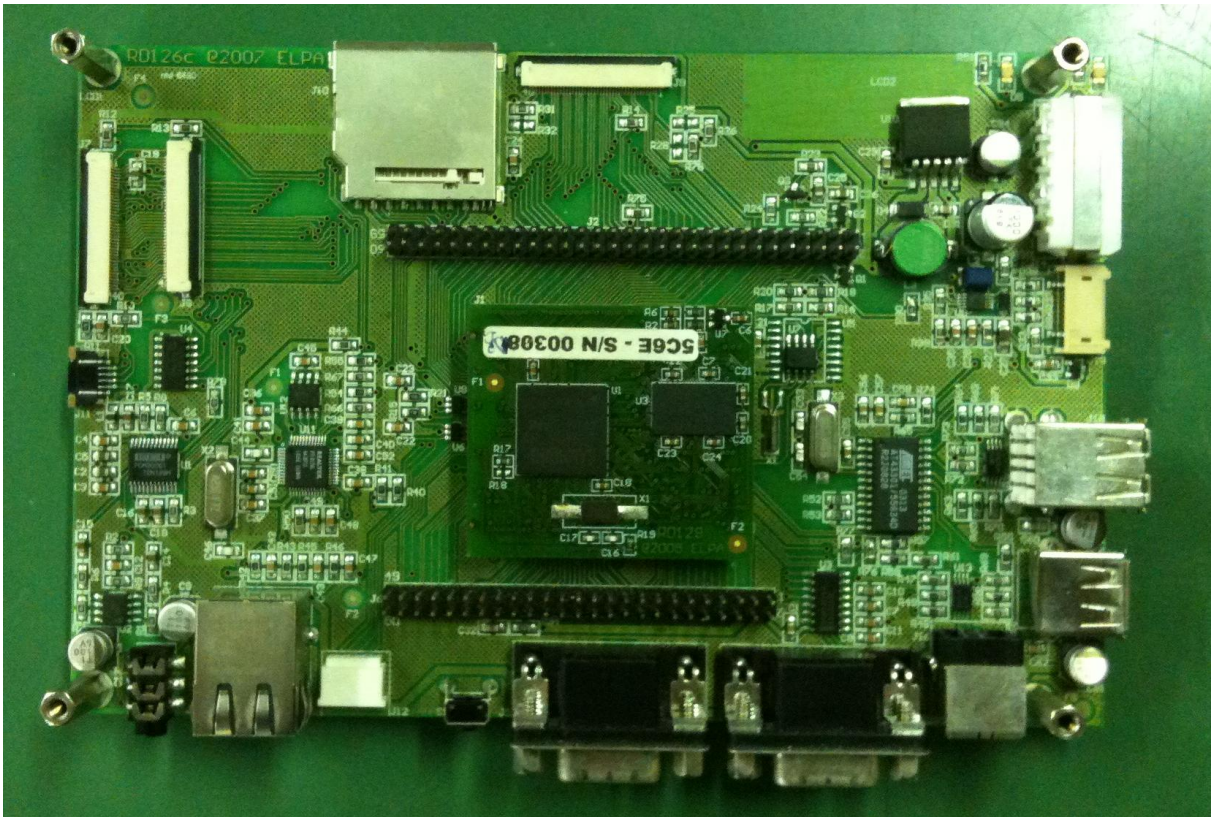


Figure 188: RD126 development board for RD129

Appendix F: StarSim Core features

The StarSim Core is based on a Work Station with the following characteristics:

Processor: Intel Xeon E5-2620

MotherBoard: Asus P9 X79 ws

RAM: 16GB DDR3 ECC

HardDisk: Toshiba 1TB 7200rpm x2

SSD: Samsung SSD Pro 128GB

Professional graphic card: Nvidia Quadro 4000

I/O ports:

12 USB ports (2x 3.0)

2 serial ports

2 ethernet ports

DVD burner

Operating System: Linux Real Time

Riassunto

Negli ultimi decenni, la complessità dei sistemi è aumentata in modo considerevole. Sono aumentati esponenzialmente sia il numero di funzioni che possono essere eseguite da un singolo sistema, grazie al enorme balzo tecnologico, sia la quantità di relazioni tra funzioni e hardware nonché le interazioni di elementi e discipline che concorrono alla definizione del sistema stesso. La crescente complessità sottolinea l'importanza di definire metodi e strumenti che migliorino la progettazione, la verifica e la validazione del sistema: efficienza e riduzione dei costi senza la perdita di affidabilità e di confidenza nel prodotto finale sono gli obiettivi da perseguire.

Nel campo del System Engineering, il moderno approccio Model and Simulation-based sembra rappresentare una strategia promettente per il raggiungimento degli obiettivi, perché è in grado di permettere la riduzione delle risorse sprecate rispetto ai metodi tradizionali, oltre che il risparmio di denaro e di tempo nell'esecuzione di lavori lunghi e noiosi. Il *Model Based System Engineering (MBSE)* è una disciplina che sta prendendo piede nel campo dell'ingegneria perché rende possibile in qualsiasi momento (concordemente con la fase del progetto in corso) la fattibilità, le capacità e le prestazioni del sistema attraverso la simulazione. La simulazione viene utilizzata lungo l'intero ciclo di vita del prodotto e può variare da sessioni di simulazioni puramente numeriche (in cui il comportamento del sistema e delle condizioni ambientali e di funzionamento è completamente riprodotto da modelli virtuali) a simulazione con il software e l'hardware di bordo completamente integrati nelle reali condizioni ambientali. All'interno di questo range, è quindi possibile definire differenti stadi di simulazione intermedi: *algorithm in the loop (AIL)*, *software in the loop (SIL)*, *controller in the loop (CIL)*, *hardware in the loop (HIL)*, ed ulteriori configurazioni ibride tra queste.

L'attività di ricerca di questa tesi intende definire e validare una metodologia iterativa (basata appunto su modelli e simulazioni) in grado di fornire supporto al team di ingegneria con l'ambizione di migliorare l'efficacia del progetto e della verifica di un sistema spaziale, con un particolare interesse verso il sottosistema di Guida Navigazione e Controllo (GNC). La scelta di concentrarsi sul GNC deriva dagli interessi comuni e dal background dei gruppi coinvolti in questo programma di ricerca: ASSET (AeroSpace System Engineering Team) del Politecnico di Torino e AvioSpace s.r.l., una società del gruppo EADS. Il sistema di GNC è sufficientemente complesso (e richiede sia conoscenze specialistiche sia competenze tipiche dell'ingegnere di sistema), di vitale importanza per qualsiasi veicolo spaziale e, soprattutto, la verifica del suo funzionamento risulta molto difficile a terra, a causa delle notevoli limitazioni sulla capacità di riprodurre contemporaneamente e per lunghi periodi la dinamica di un veicolo orbitante ed i fenomeni dell'ambiente operativo in orbita.

Considerando che il processo di verifica deve essere eseguito lungo l'intero ciclo di vita del prodotto, è fondamentale avere a disposizione uno strumento, individuato in un *simulatore*, che permetta il progetto e la verifica attraverso la simulazione, indipendentemente dal grado di complessità dell'oggetto, del test da performare e della fase progettuale in corso. Buona parte del lavoro di tesi è incentrata sulla progettazione del simulatore, chiamato *StarSim*, che rappresenta il vero cuore pulsante della metodologia proposta. *StarSim* è stato interamente progettato e sviluppato all'interno di questo lavoro: dalla definizione dei requisiti all'integrazione hardware e implementazione software, fino all'assemblaggio e integrazione delle varie parti e alla verifica e validazione che ha portato al completamento della prima versione. *StarSim* è una piattaforma stand-alone al fine di ridurre i rischi di incompatibilità e perdita di informazioni che possono insorgere utilizzando diversi software e/o strumenti durante le diverse fasi del ciclo di vita, e in particolare nel passaggio da una fase alla successiva. Modularità, flessibilità, velocità, connettività, funzionamento in tempo reale, facilità di gestione dati, efficacia e congruenza degli output rispetto agli input sono le caratteristiche ricercate nella progettazione di *StarSim*. Ad ogni iterazione della metodologia, *StarSim* garantisce la possibilità di verificare il comportamento del sistema da testare grazie alla disponibilità di modelli virtuali, che

sostituiscono tutti quegli elementi non ancora disponibili, oltre che tutte le dinamiche e le condizioni ambientali non riproducibili. Per favorire il proprio impiego, StarSim fornisce un database fornito e “user-friendly” di modelli e interfacce, in grado di coprire diversi livelli di dettaglio e fedeltà; inoltre viene supportata la possibilità di aggiornamento del database, in modo tale da consentire all'utente la possibilità di creare modelli personalizzati (rispettando poche e semplici regole di base). Progressivamente, parti del software e dell'hardware di bordo possono (e devono) essere introdotti, senza che tuttavia sia mai necessario interrompere il processo di progettazione e verifica, evitando ritardi e dispendio inutile di risorse.

StarSim è stato usato per la prima volta all'interno del programma e-st@r: si tratta di un progetto educativo mirato alla realizzazione di satelliti appartenenti allo standard CubeSat che coinvolge gli studenti del CubeSat Team del Politecnico di Torino e i ricercatori di ASSET. In e-st@r, StarSim è stato principalmente impiegato per lo sviluppo del payload, cioè un sistema attivo di determinazione e controllo dell'assetto (A-ADCS), ma in conclusione è stato anche aggiornato ed utilizzato per valutare le funzionalità, i modi operativi e le prestazioni dell'intero prodotto (il CubeSat). Lungo l'intera attività del programma, sono state effettuate simulazioni AIL, SIL, CIL e HIL, verificando con successo un gran numero di requisiti funzionali e operativi. In particolare, sono stati selezionati e verificati gli algoritmi di determinazione dell'assetto, le leggi di controllo e le modalità di funzionamento dell'A-ADCS; il software di bordo è stato sviluppato passo dopo passo sul simulatore e i file eseguibili con la versione finale, privi di banchi, sono stati caricati sul microcontroller. Tutte le interfacce e protocolli e la gestione dei dati e dei comandi sono stati verificati. Gli attuatori, i circuiti elettrici e logici sono stati progettati, costruiti e testati, ed è stata eseguita la calibrazione dei sensori e degli stessi attuatori. Problemi relativi alla sincronizzazione, alla gestione dei dati a bordo e all'esecuzione in tempo reale sono stati risolti proprio grazie alle simulazioni con StarSim. Infine sono state eseguite simulazioni hardware in the loop sia con il solo sottosistema A-ADCS sia con l'intero satellite, verificando il soddisfacimento di numerosi requisiti funzionali ed operativi dei CubeSat.

Il caso di studio rappresenta la prima validazione della metodologia attraverso l'impiego della prima versione di StarSim. Sono state mostrate le peculiarità della metodologia che vuole portare al miglioramento delle attività di progettazione e di verifica, punti nevralgici per aumentare il livello di confidenza nel successo di una missione spaziale.

Curriculum Vitae

Fabrizio Stesina was born in Biella, Italy, on May 21, 1980. He attended secondary school at the scientific institute Amedeo Avogadro, in Cossato. He received the MSc degree in Computer Engineering from Politecnico di Torino, Torino, Italy, in 2004. From 2004 to 2010, he was Research Assistant in the Department of Mechanical and Aerospace Engineering of Politecnico di Torino. In January 2011. He started the PhD degree in Aerospace Engineering at Politecnico di Torino, Torino, Italy.

His main scientific interests are: Guidance, Navigation and Control Systems for space applications, System Engineering, Control Theory, and Techniques for systems simulation and verification.

He has been working on small satellite programs carried out within the Aerospace Systems Engineering Team, ASSET, and he has the role of System Engineer in CubeSat student projects. He has also been participating in several research activities in the aerospace engineering field mainly sponsored by ESA (European Space Agency), MIUR (Ministero dell'Istruzione, Università e Ricerca) and Piedmont Regional Governement.