Title	Design of Reusable Software for Attitude Determination System of Micro/Nano Satellites in Consideration of Modularity and Extensibility using SysML
Sub Title	
Author	Nguyen, Son Duong(Nishimura, Hidekazu) 西村, 秀和
Publisher	慶應義塾大学大学院システムデザイン・マネジメント研究科
Publication year	2015
Jtitle	
JaLC DOI	
Abstract	
Notes	修士学位論文. 2015年度システムエンジニアリング学 第179号
Genre	Thesis or Dissertation
URL	https://koara.lib.keio.ac.jp/xoonips/modules/xoonips/detail.php?koara_id=KO40002001-00002015-0007

Master's Dissertation 2015

Design of Reusable Software for Attitude Determination System of Micro/Nano Satellites in Consideration of Modularity and Extensibility using SysML

Nguyen Son Duong
(Student ID Number: 81334561)

Supervisor: Prof. Hidekazu Nishimura

September 2015

Graduate School of System Design and Management,
Keio University
Major in System Design and Management

SUMMARY OF MASTER'S DISSERTATION

Student Identification Number	81334561	Name	Nguyen Son DUONG

Title

Design of Reusable Software for Attitude Determination System of Micro/Nano Satellites in Consideration of Modularity and Extensibility using SysML

Abstract

Today, micro and nano satellites are being widely used for Earth observation, remote sensing, technology demonstration and scientific purposes with lower cost and shorter time development in compare with larger satellites. The micro/nano satellites projects are usually varied according to missions and hardware constraints. Among on-board software components, attitude determination software is one of the most difficult parts when coding because it requires complicated calculations, especially for the micro/nano satellites. The ratio of the attitude determination software can be estimated about 40% in total on-board software of satellites. Therefore, design of the reusable attitude determination software which can be reused for many types of the micro/nano satellite projects to save time and cost for satellite development is the objective of this research.

NASA has already realized software reusability viewpoint in NASA Reuse Readiness Levels (RRLs). NASA RRLs is a method to measure the potential of reuse of general software and has 9 topic areas including documentation, modularity, extensibility, etc. After analyzing the context of attitude determination software reuse between micro/nano satellites projects, the modularity and extensibility are critical and therefore selected in this research.

Model-based systems engineering (MBSE) is a powerful approach to model complex systems such as the satellites. Systems Modeling Language (SysML) is a standardized language to enable MBSE. Applying SysML to design the satellite and the on-board software has outstanding benefits and is now very promising. By using SysML, not only the software itself is designed but also the whole system can be modeled as well. This point is very important when reuse software because the attitude determination software has constraints and relationships with other systems of the satellite. Besides, by using SysML, not only the modularity and extensibility but also the documentation is well supported.

In brief, by using SysML and utilizing the viewpoint of modularity and extensibility from NASA RRLs, the highly modular and extensible attitude determination software has been designed in this research in order to maximize the potential of reuse for micro/nano satellites. For future works, the necessary of applying SysML for designing satellites should be more persuaded to the satellite developers.

Key Words (5 words): SysML, micro/nano satellites, attitude determination software, design reusable software, NASA RRLs.

Acknowledgements

I would like to thank my supervisor, Prof. Nishimura Hidekazu, without his supports and guidance during my 2 years at Keio SDM, this thesis could not be finished. He has provided us many professional software tools to practice SysML such as Rhapsody and Cameo Enterprise Architecture. I will apply knowledge about model-based system engineering (MBSE) and SysML learning from him to design satellites when I come back to my country.

I would like to thank Prof. Seiko Shirasaka for his regular meetings to improve the quality of my research. I also would like to thank Prof. Makoto Ioki for his useful advices and comments to my thesis.

I also would like to thank Shusaku Yamaura-sensei, Takashi Hiramatsu-sensei and Ayumu Tokaji-sensei, who have continuously help me to do this research as well as to finish my study at Keio SDM.

Lastly, I would like to thank my organization, Vietnam National Satellite Center, and my family who gave me a great deal of encouragement.

Table of Contents

I. Introduction

I.1. Problem background

The weight of micro satellites is in the range of 10-100 kg and that of nano satellites is in the range of 2-10 kg [1]. The micro and nano satellites are now becoming a new trend. There are more and more companies and organizations who are interested in developing these types of satellites [1], [2]. In 2014, there were 158 micro/nano satellites launched globally, this showed an increase of nearly 72% compared to 2013 [3]. The characteristics of micro/nano satellites are low cost, short time development, varied on mission requirements and limited on calculation capability compare with larger satellites.

Software coding is always the difficult task in satellite development because it requires a lot of time for developing and testing, especially for the micro/nano satellite projects. For each project, developers need to design and implement software for satellite's on-board computer including attitude determination software. The attitude determination software is one of the most complicated parts of the on-board software because it involves complex calculations [4] [5].

In the Table I.1.1, the estimating software components of a typical satellite are showed [6]. Form the Table I.1.1, the software components related to the attitude determination and control are the main part of the satellite on-board software, and nearly 40% is the attitude determination software. Therefore, development of the attitude determination software is more expensive and time consuming than other software components.

Table I.1.1. Estimating source lines of code for typical satellite functions

Software Component	Source Lines of Code
Executive	1,000
Communication	5,500
Attitude/Orbit Sensors Processing	8,100
Attitude Determination and Control Kalman Filter	29,800-33,800 6,000-10,000
Attitude Actuator Processing	3,900
Mathematic Utilities	5,700
Fault Detection	11,500
Other Functions	5,000

Software reuse for the attitude determination system is expected as a solution to save time and cost of satellite development, especially for organizations like Vietnam National Satellite Center (VNSC), who need to develop not only one but several micro/nano satellites for different type of applications. Software reuse can help save time and cost for coding and testing as well as increase software reliability.

However, according to Ref. [7], research on on-board software reusability is still low and its realization is urgently desired. The main reasons are the difficulties of software reuse in terms of the difference in CPU type, interfaces and mission requirement of the micro/nano satellites.

The constraints which lead to the difficulties of software reuse between micro and nano satellite projects can be showed in an example in Table I.1.2. As can be shown in the Table I.1.2, each of the satellite projects has the different mission as well as the determination accuracy requirement, the calculation performance of the on-board computer and the model of used sensors. Therefore, in order to be reused, the attitude determination software need to be designed to deal with the variety of each satellite projects. For example, it is difficult to reuse the attitude determination software from the first satellite project (Micro Dragon) to the second satellite project (Nano satellite) because the second satellite does not have the Star Tracker as the first satellite and the performance of the on-board computer is lower.

Table I.1.2. Example of the different mission requirements and hardware constraints between micro/nano satellites projects

Satellite Projects	Mission	Determination Accuracy	On-board Computer Performance	Sensors Model
MicroDragon (50 kg)	Ocean color observation	~0.05º	48 MIPS 64 MB SDRAM	1 Star Tracker 6 Sun Sensors High Accuracy Gyro
Nano Satellite (3 kg)	Communication	~3º	16 MIPS 8MB SDRAM	0 Star Tracker 3 Sun Sensors Low Accuracy Gyro
Micro Satellite (100 kg)	Astronomy	~0.003º	96 MIPS 64 MB SDRAM	2 Star Trackers 6 Sun Sensors High Accuracy Gyro

I.2. Definition of software reuse

The purpose of software reuse is to improve software quality and productivity. Software reuse is the use of existing software or software knowledge to construct new software. Reusable assets can be either reusable software or software knowledge. Reusability is a property of a software asset that indicates its probability of reuse [8]. A software artifact is any item which is created during the software development life cycle. A software asset is an artifact which has a particular value. A software component is a clearly delineated piece of software that performs a useful function within a software system [9].

Reusable software artifacts are not limited to just code. These assets may include algorithms and models, architectures and design patterns, systems modules and scripts, technical documentation and test results, and use metrics as well as other artifacts produced during the software development life cycle [10]. Therefore, the software developers can reuse requirements documents, design structures and any other development artifact.

The potentially reusable aspects of software projects are shown in Table I.2.1 [11].

Table I.2.1. Reusable aspects of software projects

1. Architectures	6. Templates
2. Source code	7. Human Interfaces
3. Data	8. Plans
4. Designs	9. Requirements
5. Documentation	10. Test Cases

I.3. The software reusability viewpoint from NASA RRLs

In order to design highly reusable software, it is necessary to know how the reusability of software can be evaluated. This information will be utilized as a viewpoint when design reusable software.

The NASA Earth Science Data Systems – Software Reuse Working Group has been developed Reuse Readiness Levels (RRLs) since 2008 for use as a measure to evaluate the potential reusability of software. Initially developed for the Earth science domain, but it is applicable to general. The RRLs are being developed to offer capabilities for developers and reusers of software to measure the reusability of software [12].

Table I.3.1. The summary of 9 levels of software reuse according to RRLs [12]:

Level	Summary
RRL 1	Limited reusability; the software is not recommended for reuse.
RRL 2	Initial reusability; software reuse is not practical
RRL 3	Basic reusability; the software might be reusable by skilled users at substantial effort, cost, and risk.
RRL 4	Reuse is possible; the software might be reused by most users with some effort, cost, and risk.
RRL 5	Reuse is practical; the software could be reused by most users with reasonable cost and risk.
RRL 6	Software is reusable; the software can be reused by most users although there may be some cost and risk.
RRL 7	Software is highly reusable; the software can be reused by most users with minimum cost and risk.
RRL 8	Demonstrated local reusability; the software has been reused by multiple users.
RRL 9	Demonstrated extensive reusability; the software is being reused by many classes of users over a wide range of systems.

Using NASA RRLs to measure software reusability has many advantages. RRLs can be used by different types of stakeholders in different situations. Software can be evaluated either by using the RRLs in a simple manner by the summary RRLs or more extensively by each of the nine topic areas to get a precise assessment. The project managers have a single number by the summary RRLs that is quick and easy to understand. Software developers have a simple way to quickly estimate the readiness of software assets to be reused. Although an estimation of reuse readiness will not reduce the tasks of testing candidate reusable assets, it will enable software developers to more easily determine how ready the software is for their purposes. The software developers also can have more detailed information by the RRL topic area levels [12]. Software providers need to evaluate whether their software can be used by others, whether the software is ready for reuse, and which parts need to be enhanced for use by others. Software reusers need to check whether to consider reusing a software asset, compare software assets available for reuse, assess strengths and weaknesses of such software, and recognize where additional development is necessary for reuse [13].

In NASA RRLs, there are 9 topic areas to measure software reuse including: Portability, Extensibility, Documentation, Support, Packaging, Intellectual Property Issues, Standards Compliance, Verification and Testing, and Modularity.

Examples of the difference between limited reusable software and highly reusable software when comparing software by each topic area in NASA RRLs are shown in Table I.3.2 and Table I.3.3.

Table I.3.2. Comparison by Documentation, Extensibility and Modularity

	Documentation	Extensibility	Modularity
Limited reusable software	Source code is available without documentation.	Parameters cannot be changed.	Not designed with modularity
Highly reusable software	Documentation on design, customization, testing, use, and reuse is available.	Use configuration files. The extensibility capability for the software is well defined.	Organize software components into libraries

Table I.3.3. Comparison by Portability, Standards Compliance and Verification and Testing

	Portability	Standards Compliance	Verification and Testing
Limited reusable software	The software is not portable.	No standards compliance.	No testing performed
Highly reusable software	The software can be ported to all systems.	The software and software development process comply with internationally recognized standards.	Software application tested and validated in a relevant context

In this research, after analyzing these topic areas in NASA RRLs, the modularity and extensibility are focused in this research because they are very important for attitude determination software.

Modularity is a software design technique that increases the extent to which software is composed from separate components, called modules. Conceptually, modules represent a separation of and encapsulation of concern, purpose, and function [12]. To achieve modularity

of level 7 in NASA RRLs, in software, modules should be created for all specified functions and organized into libraries with consistent features within interfaces.

Extensibility is an important dimension to be able to incorporate an asset and add to or modify its functionality [12]. To achieve extensibility of level 6 in NASA RRLs, software should be designed to allow extensibility across a moderate to broad range of application contexts, provides many points of extensibility.

I.4. The benefits of applying MBSE and SysML to design software for complex systems

Traditionally, large projects have employed a document-based system engineering approach. However, the document-based approach has some critical limitations. The completeness, consistency, and relationships between requirements, design, engineering analysis, and test information are hard to assess because this information is spread across several documents. To understand a particular aspect of the system and to perform the necessary traceability and change impact assessments are also difficult and leads to poor synchronization between system-level requirements and lower-level hardware and software design. As a result, to reuse the system requirements and design information for the design of the new systems is limited [14].

Compare with document-based system engineering, model-based systems engineering (MBSE) is proved as a more effective approach to manage complexity, improve design quality and communication between developers. MBSE applies systems modeling as part of the system engineering process to support analysis, specification, design, and verification of the developed system [14].

Systems Modeling Language (SysML) is a standardized language to enable MBSE. SysML is an extension of the Unified Modeling Language (UML) version 2, which has become the standard software modeling language. SysML supports the specification, design, analysis, and verification of systems which may include not only software but also data, hardware, etc.

SysML is a graphical modeling language for representing requirements, behaviors, structure, and properties of the system and its components [14]. SysML has capability to model complex systems from a broad range of domain such as satellites. By using SysML, the design of attitude determination software is consistent with the other sub-systems of satellite. Therefore, SysML is more suitable to design software components of satellite than UML.

The type of diagrams of SysML is showed in Figure I.4.1 [14]. The requirement diagram and parametric diagram are the new diagrams compare to UML. One benefit of SysML is allowing traceability between requirements and design model.

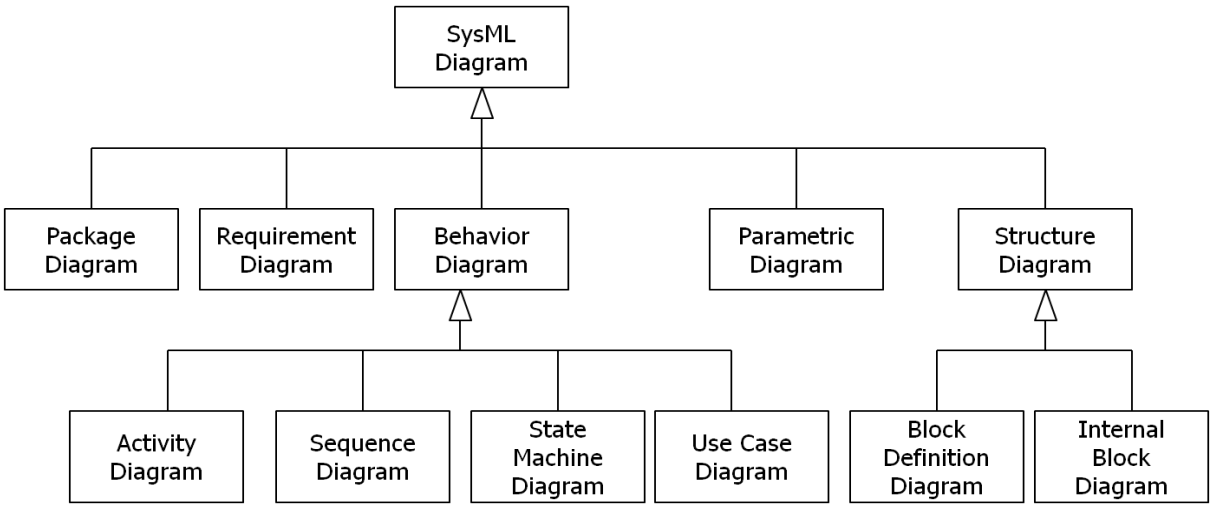

Figure I.4.1. SysML diagram taxonomy

I.5. Research objective and approach

The objective of this research is to design highly modular and extensible attitude determination software which can be reused for a variety of micro/nano satellite projects to save time and cost of satellite development.

The context of software reuse here is mainly for the software design and source code reuse. The software will be designed and implemented in C code files, so that the other programmers can utilize reusable parts in their projects. They only need to modify the changing parts for each micro/nano satellite projects. By doing so, they can save time and cost in developing attitude determination software for their satellites.

The approach of this research is to design the reusable attitude determination software by using SysML with the focus on modularity and extensibility from the viewpoint of NASA RRLs.

The modularity in my research is the degree of decomposition of the attitude determination software. There are three steps of decomposition in my research.

In the first step, based on functional analysis, the attitude determination software is decomposed into software modules such as sensor processing, reference vectors estimation, attitude estimation. The sensor processing module is to process the data outputs of sensors. The reference vectors estimation module is to calculate the reference vectors in Earth-Centered Inertial (ECI) frame. The attitude estimation module contains the algorithms for attitude estimation which will be which can be selected based on different accuracy requirements and hardware constraints of each micro/nano satellites projects.

In the second step, each software module is decomposed into components. For example, the sensor processing module is decomposed into the components correspond to each types of sensors.

In the third step of decomposition, each component is decomposed into basic functions and the standard interfaces are defined between basic functions. For example, the software component "Gyro Processing" is decomposed into two basic functions including "Get Angular Rates in Gyro Frame" and "Calculate Angular Rates in Satellite Body Frame".

The modularity is also to support the extensibility. In my research, the extensibility is the ability to change the sensors model and performance of onboard computer of satellite projects. For example, there are 6 Sun sensors, 1 Star tracker in Micro Dragon satellite project; however, there is no Star tracker in Nano satellite project, and the performance of onboard computer of Nano satellite is lower.

To achieve the extensibility in my research, there are two steps. Firstly, mission and sensor dependent parameters are separated by data store which can be easily modified when reuse. Secondly, for each satellite project, the attitude estimation algorithms are selected based on the time constraint of calculation.

I.6. Structure of the Thesis

In this research, chapter 1 discusses about the problem background, the objective and approach of the research. The research problem is how to design the reusable attitude determination software to apply for variety of micro/nano satellite projects to save time and cost of satellite development. The approach of this research is to design by using SysML and selecting viewpoint of modularity and extensibility from NASA RRLs base on analyzing the difference between micro and nano satellites projects in term of mission and hardware.

Chapter 2 overviews about the attitude determination and control system of satellite including the attitude determination system and the constraints of design reusable attitude determination software for micro/nano satellites. The attitude determination system is one of the system of the attitude determination and control system. Therefore, the design of the reusable attitude determination software for attitude determination system need to consider the attitude determination and control system of satellites in general as well as the design constraints of the reusable attitude determination software.

Chapter 3 shows the detail design of reusable attitude determination software for micro/nano satellites by using SysML including the design of each software modules and components. The modularity and extensibility also are explained in this chapter.

Chapter 4 shows the verification and validation of this research. The verification is done by both SysML models and calculation the ratio of software reuse. The validation is done by analyzing the situation of reuse attitude determination software between Micro

Dragon and Nano satellite projects. An interview to check whether the purpose of saving time and cost by using this design is also established.

Chapter 5 is the conclusion to summarize the results of this research and discussions about the future works including the necessary of applying SysML for designing satellites.

II. Overview of attitude determination system and constraints of design reusable attitude determination software

II.1. Overview of attitude determination and control system of satellite

The main satellite subsystems are attitude determination and control system (ADCS), communication, thermal control, power, and command & data handling.

ADCS is one of the most important subsystems of satellite because it helps satellite achieve its mission such as pointing camera to take picture, solar panel direct to the Sun for battery charging.

The ADCS has three subsystems: attitude determination system (ADS), attitude control system (ACS) and attitude guidance. The function of ADS is to determine the current attitude and position of the satellite. The current attitude will be the input of the ACS. The ACS will compare the current attitude and the desired attitude to calculate the required torque for actuators.

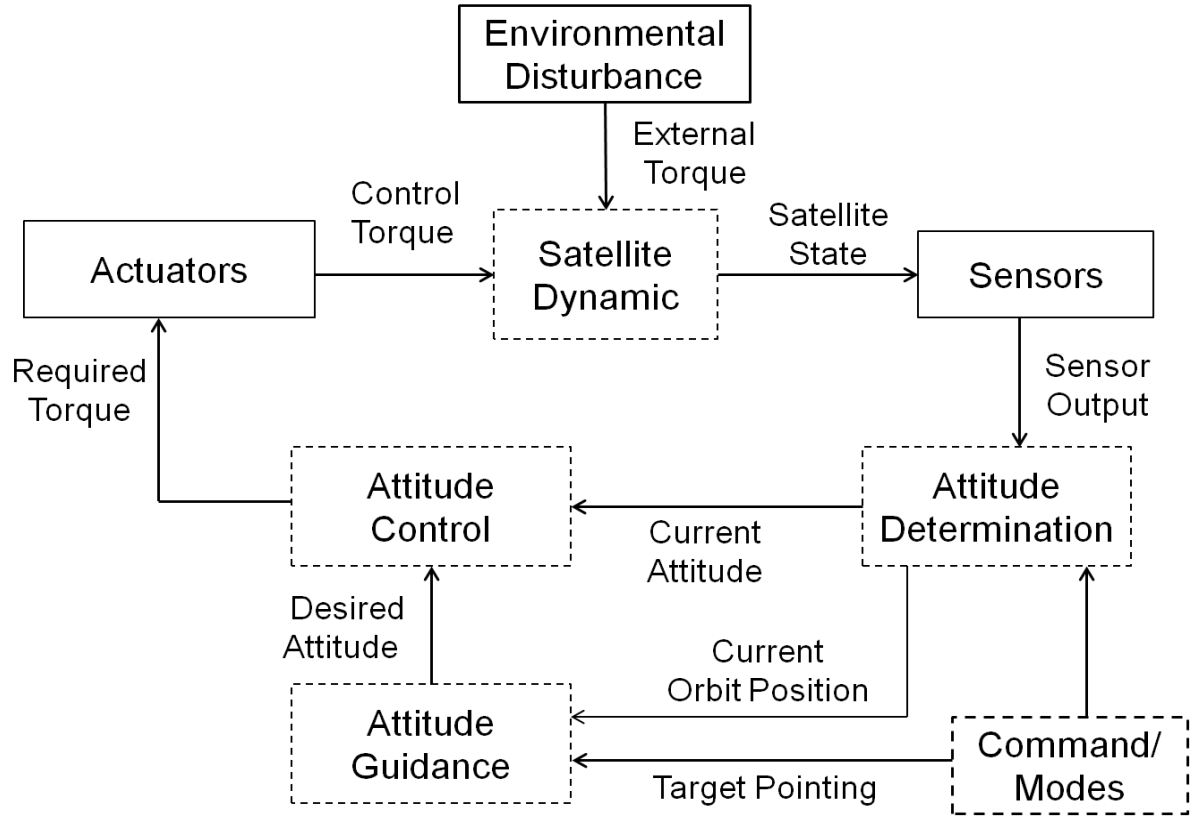

Figure II.1.1. The block diagram of ADCS

In terms of the hardware, the ADCS consists of sensors, onboard computer and actuators. For the ADS, the hardware only consists of sensors and onboard computer. The usual sensors used for ADS including Sun sensors, magnetometer, gyro, star tracker and GPS receiver.

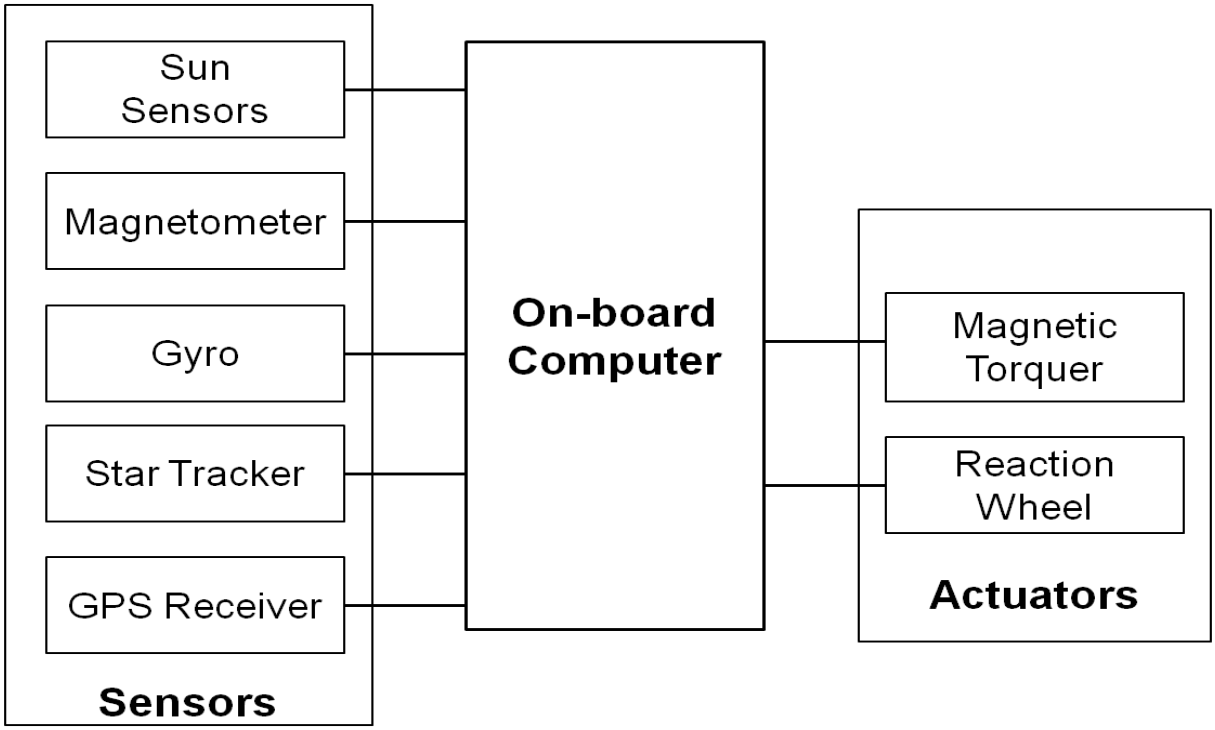

Figure II.1.2. The hardware diagram of ADCS

ADCS software was usually developed for each individual satellite project. The ADCS software including ADS software will be embedded software which runs on the onboard computer.

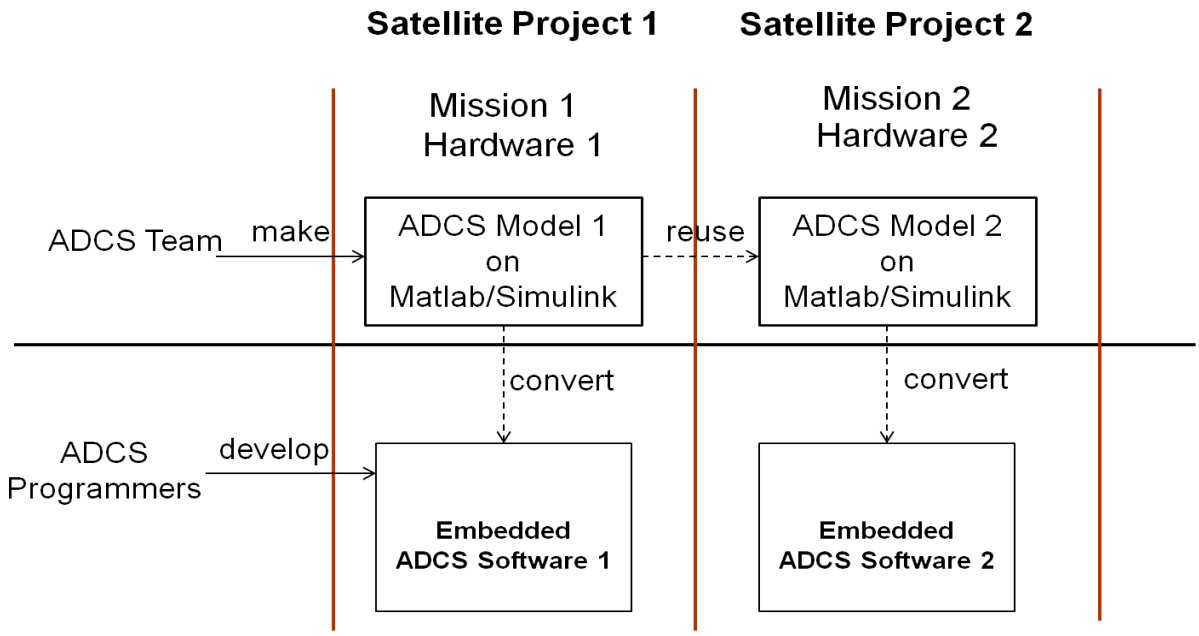

Figure II.1.3. The development of ADCS software between satellite projects

II.2. The modes of attitude determination

Each satellite has the own modes for attitude determination and control system. The name of each mode can be different from satellites. However, the purpose of each mode should be the similar. For each mode, the attitude determination should be defined in order to consider about the reuse of attitude determination software between satellites.

The modes of attitude determination and control of satellite and the explanation of the output of attitude determination for each mode are following:

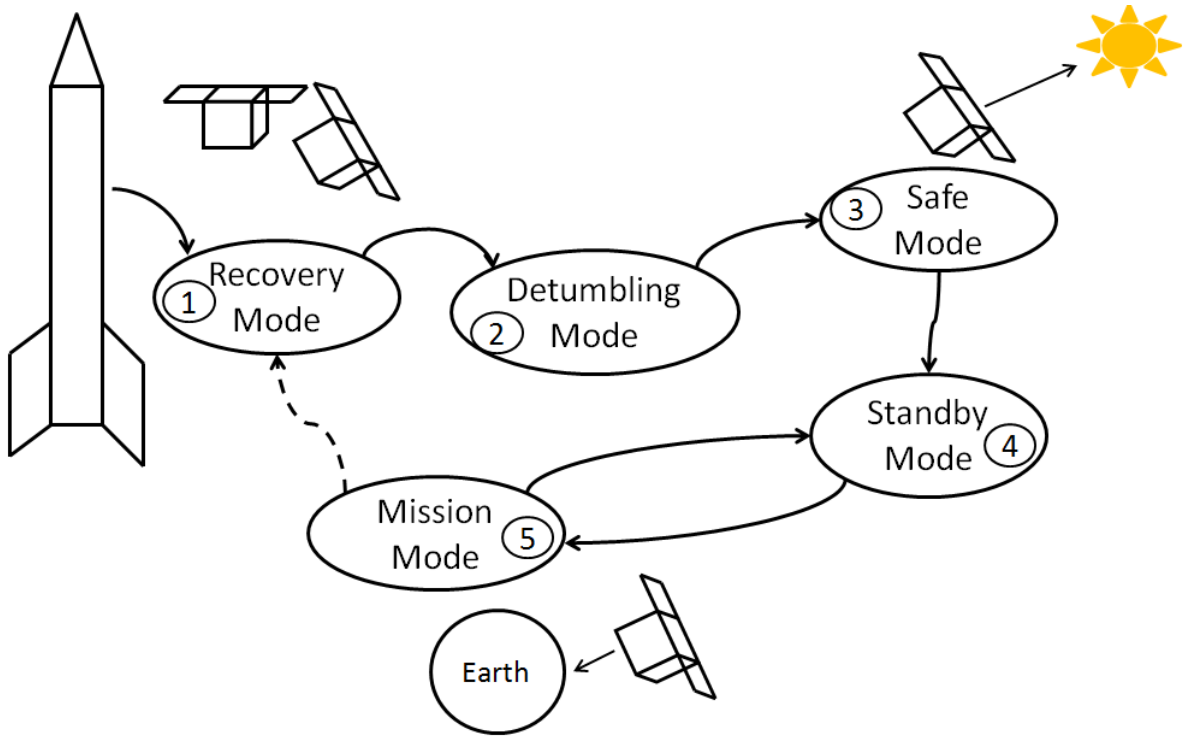

Figure II.2.1. The modes of attitude determination and control of satellite

1. Recovery mode

This mode begins right after the satellite is separated from rocket. At this moment, almost all satellite's components are turn off except the communication system to save energy for survival of satellite. This is also the first time satellite communicates with ground station by sending the telemetries and receiving the commands.

At this mode, there is no attitude determination because all the satellite's sensors and actuators are turn off.

2. Detumbling mode

This is the second mode of satellite operation following the recovery mode. At this mode, the satellite is rotating very fast. Therefore, the purpose of this mode is to make the satellite rotate slowly.

At this mode, only the magnetometer is turn on. This sensor measure the B-Field vectors on satellite body frame.

3. Safe mode (Sun pointing mode)

This is the first time the satellite gets energy from the Sun. There are two kinds of Sun pointing mode including:

+ Spin Sun pointing: the satellite does not need to turn on the reaction wheels, only need to turn on the magnetometer and the Sun sensors.

+ 3axis Sun pointing: the satellite need to turn on the reaction wheels, the control algorithm is PD. Therefore, the attitude should be the quaternion output from the attitude determination function.

In order to save battery of the satellite, in this research, the Sun pointing mode should be Spin Sun pointing, therefore only the magnetometer and Sun sensors are turn on.

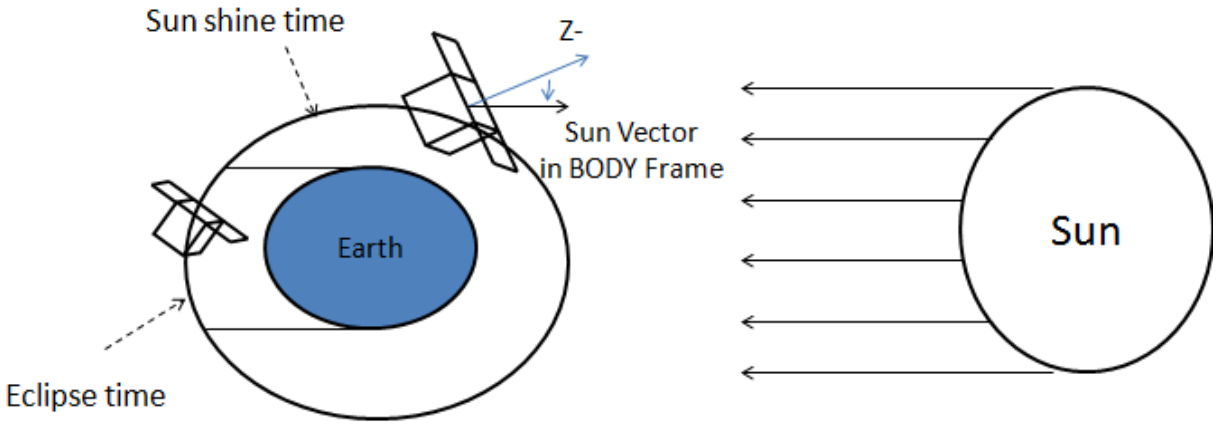

Figure II.2.2. Attitude determination at Sun Pointing mode

4. Standby mode (Idle mode)

The purpose of this mode is to save energy. The satellite faces to the Sun or to the Earth depend on the design.

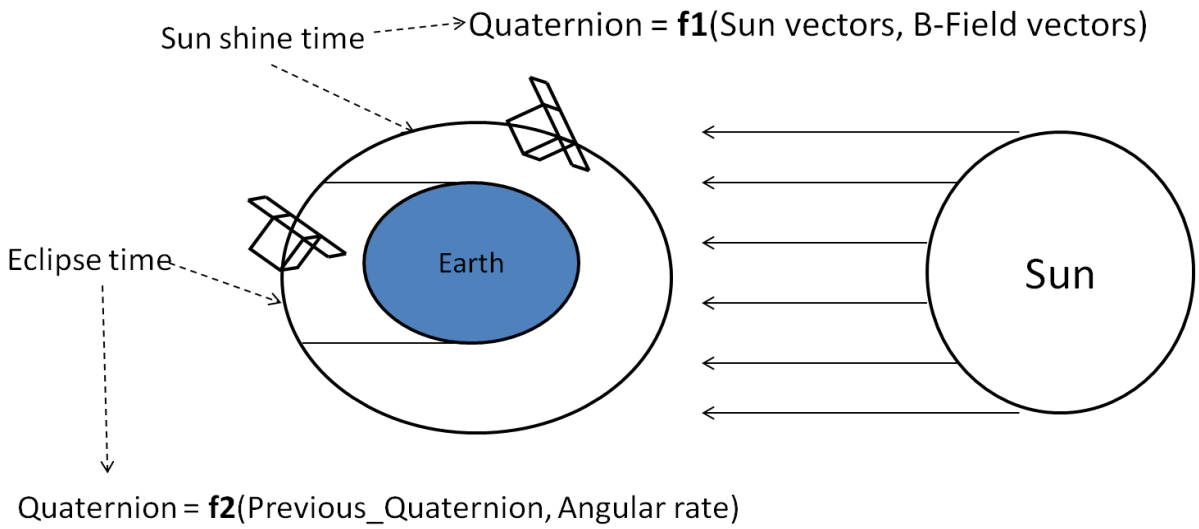

Figure II.2.3. Attitude determination at Standby mode

5. Mission mode

At this mode, depend on the mission of each satellite projects, satellite controls its camera to take images of the Earth by nadir pointing or target pointing; or moving its antenna for communication with other satellites. At this mode, almost all sensors are turn on and the attitude determination is similar to the Standby mode. However, the star tracker is used in case of high accuracy pointing.

The considering of the attitude determination for each mode is the application of the modular design in this research. The attitude determination algorithms should be designed for each mode of satellite depend on the constraint of satellite's power consumption, the accuracy requirements and the availability of the sensors. In brief, the outputs of attitude determination function for each mode are defined in the table II.1.

Table II.2.1. The output of attitude determination function of each mode

Modes	Sensors					Output of Attitude Determination
	Sun Sensor	Magnetometer	Gyro	Star Sensor	GPS Receiver	
Recovery	OFF	OFF	OFF	OFF	OFF	No Attitude Determination
Detumbling	OFF	ON	OFF	OFF	OFF	B-Field Vector in Satellite's BODY Frame from Magnetometer
Sun Pointing (Safe Mode)	ON	ON	OFF	OFF	OFF	Sun Vector in Satellite's BODY Frame
Standby	ON	ON	ON	OFF	ON	Quaternion
Mission	ON	ON	ON	ON/ OFF	ON	Quaternion

II.3.The constraints of design reusable attitude determination software for micro/nano satellites

II.3.1.The constraints of onboard computers

The constraints of onboard computers including calculation performance, size of memory storage and development environment.

Table II.3.1. The constraints of onboard computers

Onboard computers	Constraints
Calculation Performance	Processing speed should be feasible for reuse attitude determination algorithms
Size of memory storage	Memory for working area of reusable functions and models should be enough
Development Environment	The difference in programming language

II.3.2.The constraints of sensors

The constraints of sensors are showed in the Table II.3.2.

Table II.3.2. The constraints of sensors

Sensors	Power Consumption	Update Rate	Availability	Accuracy	Formats of outputs
Sun Sensor	Low	Always	Only in sun shine time	Low (~1 deg)	Depend on driver
Magnetometer	Low	Always	Continuous	Low (2-5 deg)	Depend on driver
Gyro	Mid/High	~20 Hz	Continuous	Accumulative Error	Depend on driver
Star Tracker	High	~1 Hz	Depend on image of stars	High	Depend on driver
GPS Receiver	Low	~1 Hz	Continuous	Mid/Low	Depend on driver

II.3.3.The constraints of satellite mission

Satellite missions will effect to design constraints including the determination accuracy requirements, the sensor models for attitude determination and the mounting locations of sensors.

III. Design of reusable attitude determination software for micro/nano satellites using SysML

III.1. Software architecture

In this design the Satellite Domain is developed to show the relationship between satellite and the external environment.

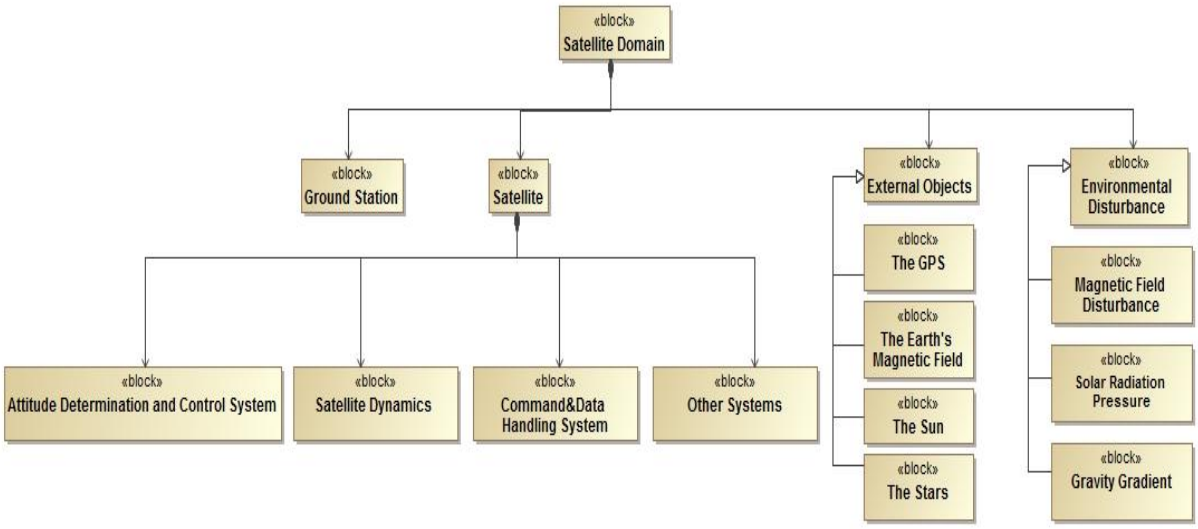

Figure III.1.1. The block definition diagram of "Satellite Domain"

The activity of control orientation of satellite for pointing to the target is showed in Figure III.1.2.

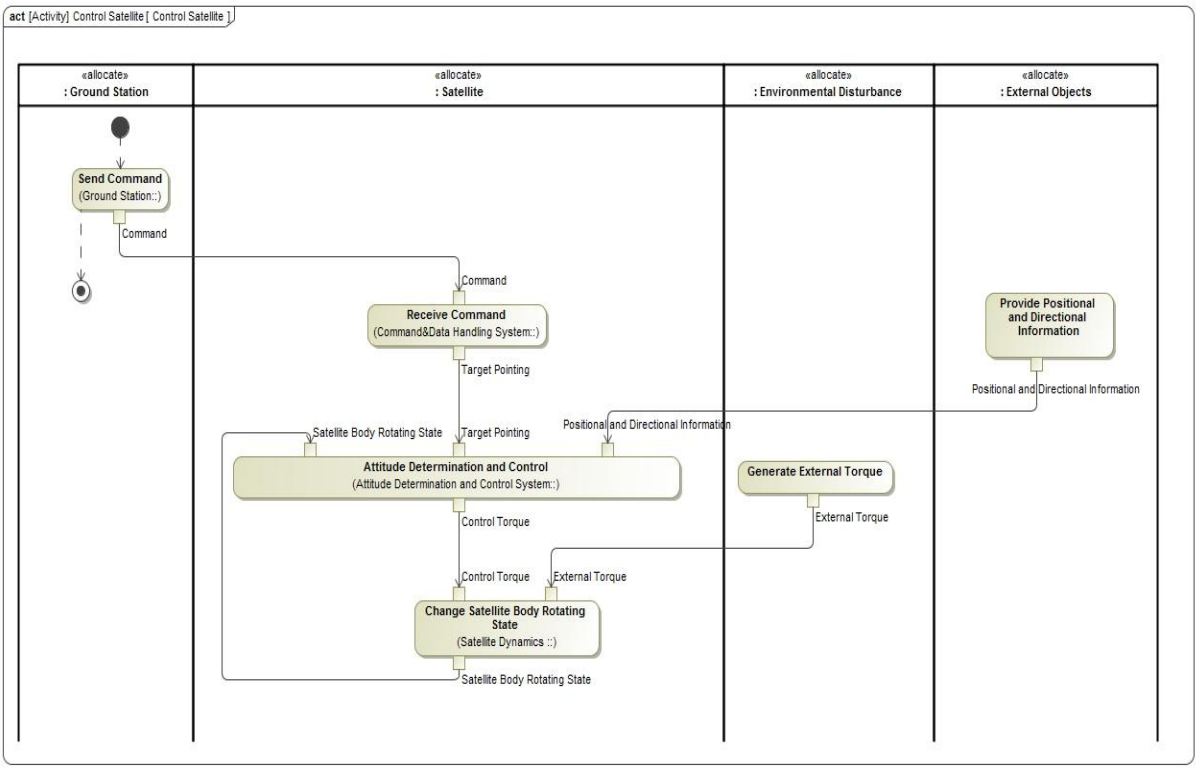

Figure III.1.2. The activity diagram of "Control Satellite"

As described in the Chapter 2, the structure of Attitude Determination and Control System is showed in Figure III.1.3.

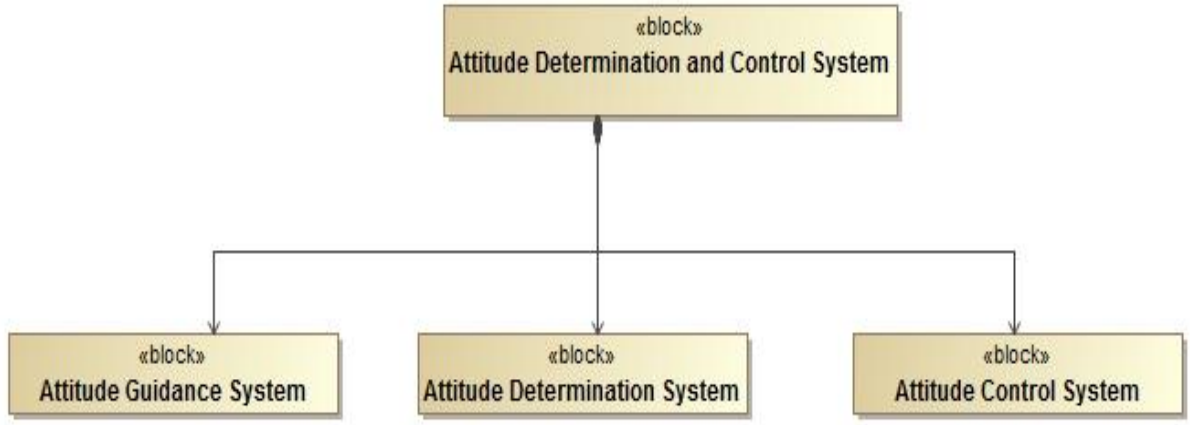

Figure III.1.3. The structure of Attitude Determination and Control System

The main function of Attitude Determination System is to estimate the current attitude of the satellite from measuring the state of satellite body rotation and processing the positional and directional information provided from external objects.

After estimated, the current attitude of satellite and the desired attitude will become the inputs for the Attitude Control System to calculate the control torque.

The activity diagram "Attitude Determination and Control" is showed in Figure III.1.4.

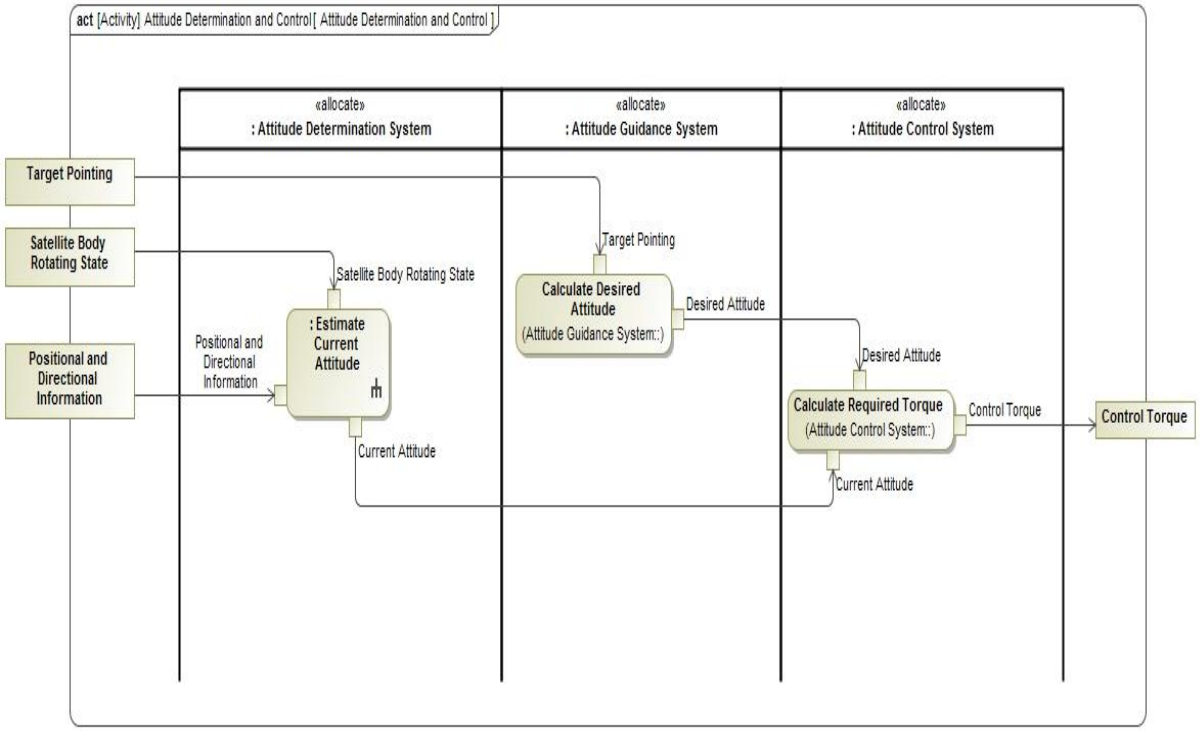

Figure III.1.4. The activity diagram of "Attitude Determination and Control"

The Attitude Determination System consists of the Attitude Determination Software and the Attitude Determination Hardware. The Attitude Determination Hardware consist of the Onboard Computer and Sensors including Gyro, Star Tracker, GPS Receiver, Magnetometer and Sun Sensor as mentioned in Chapter 2.

The block definition diagram of "Attitude Determination System"

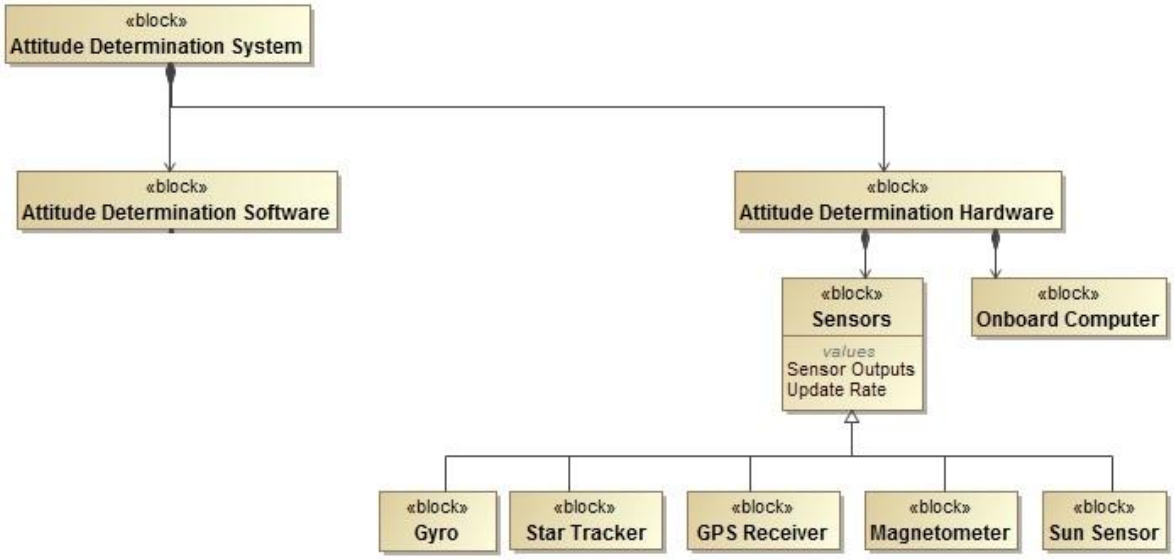

Figure III.1.5. The block definition diagram of "Attitude Determination System"

The functions of attitude determination software are analyzed from the use case diagram. The main use case of Attitude Determination Software is "Determine Attitude of Satellite" in which the actor "Attitude Determination Software" interacts with the other actors including "External Objects", "Satellite Dynamics" and "Models in Reference Frame" to determine the current attitude of the satellite.

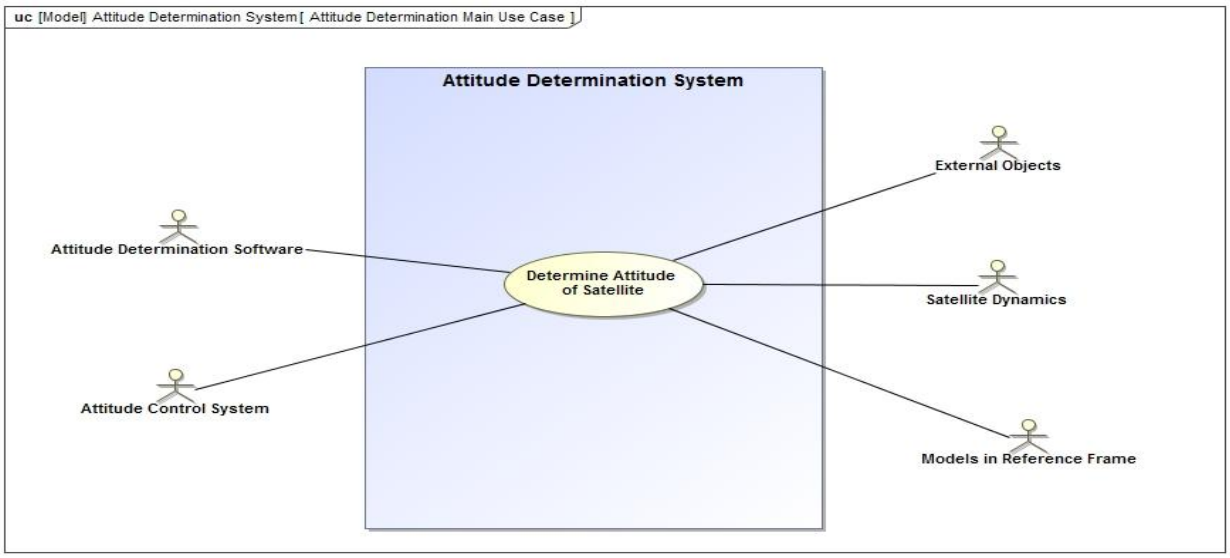

Figure III.1.6. The main use case of Attitude Determination Software

The External Objects including the GPS, the Earth's Magnetic Field, the Sun and the Stars provide the positional and directional information to the satellite.

The sequence diagram of the use case "Determine Attitude of Satellite"

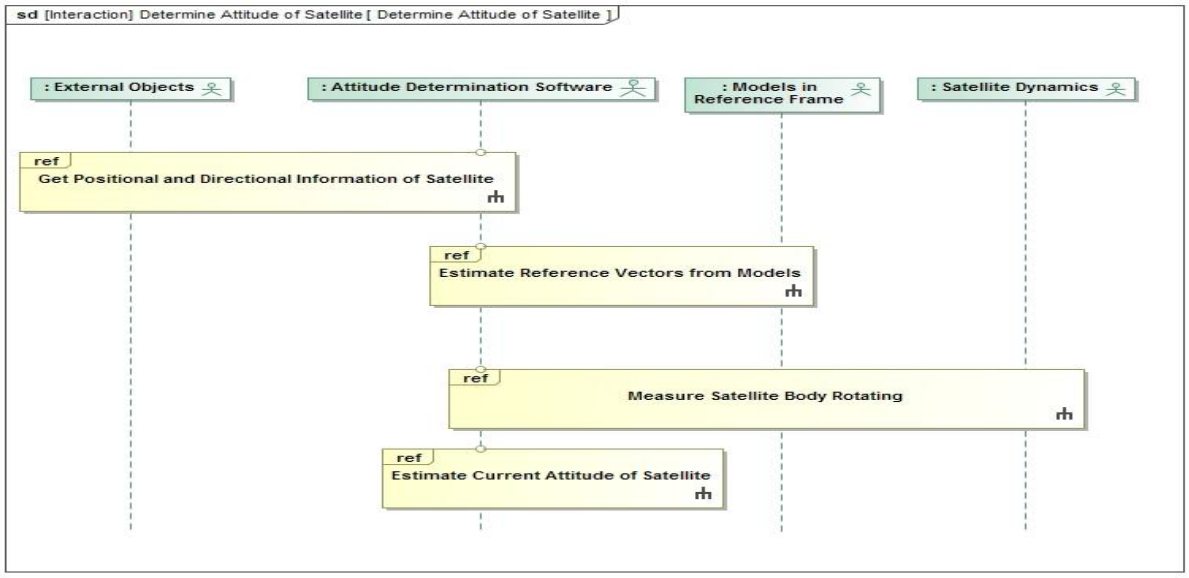

Figure III.1.7. The sequence diagram of the use case "Determine Attitude of Satellite"

The sequence diagram of "Get Positional and Directional Information of Satellite"

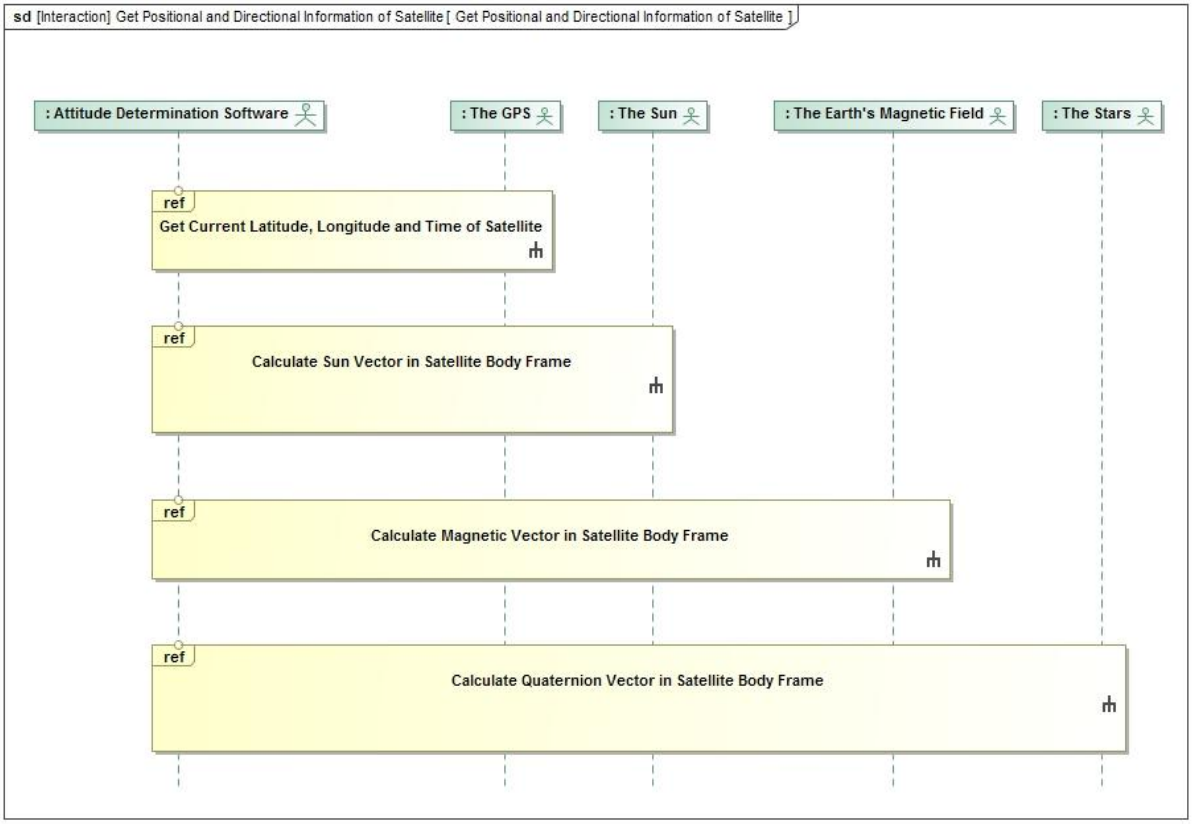

Figure III.1.8. The sequence diagram of "Get Positional and Directional Information of Satellite"

The sequence diagram of “Estimate Reference Vectors from Models”

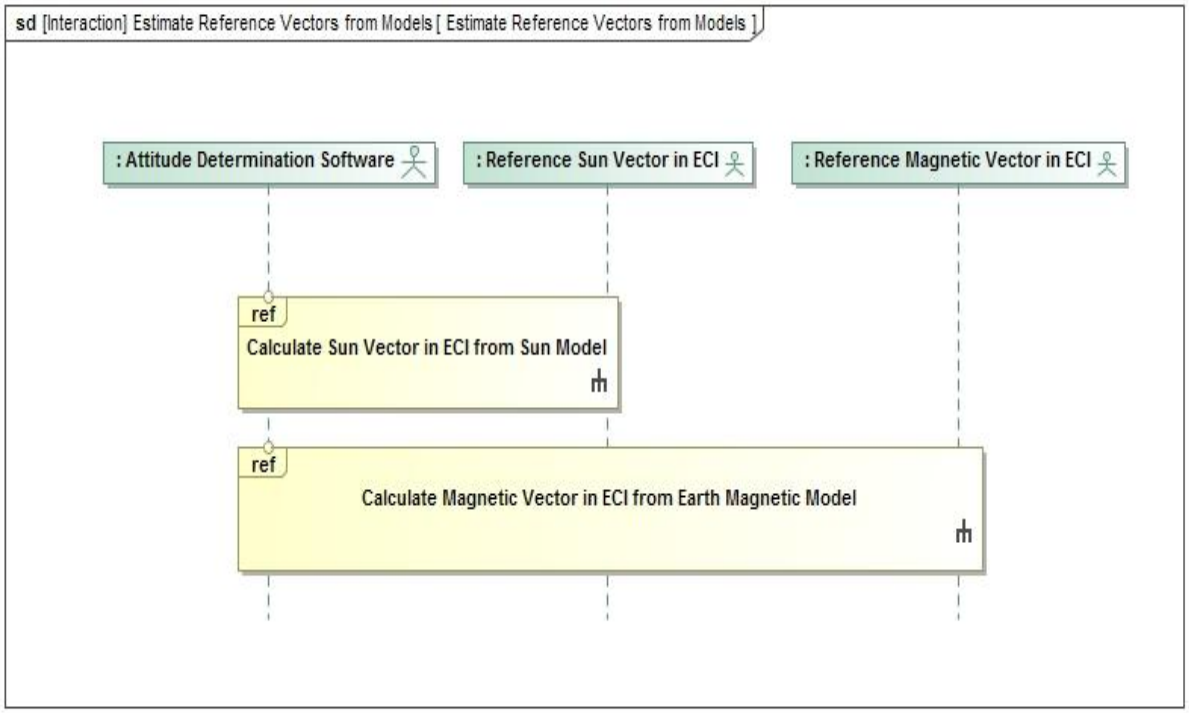

Figure III.1.9. The sequence diagram of “Estimate Reference Vectors from Models”

The sequence diagram of “Measure Satellite Body Rotating”

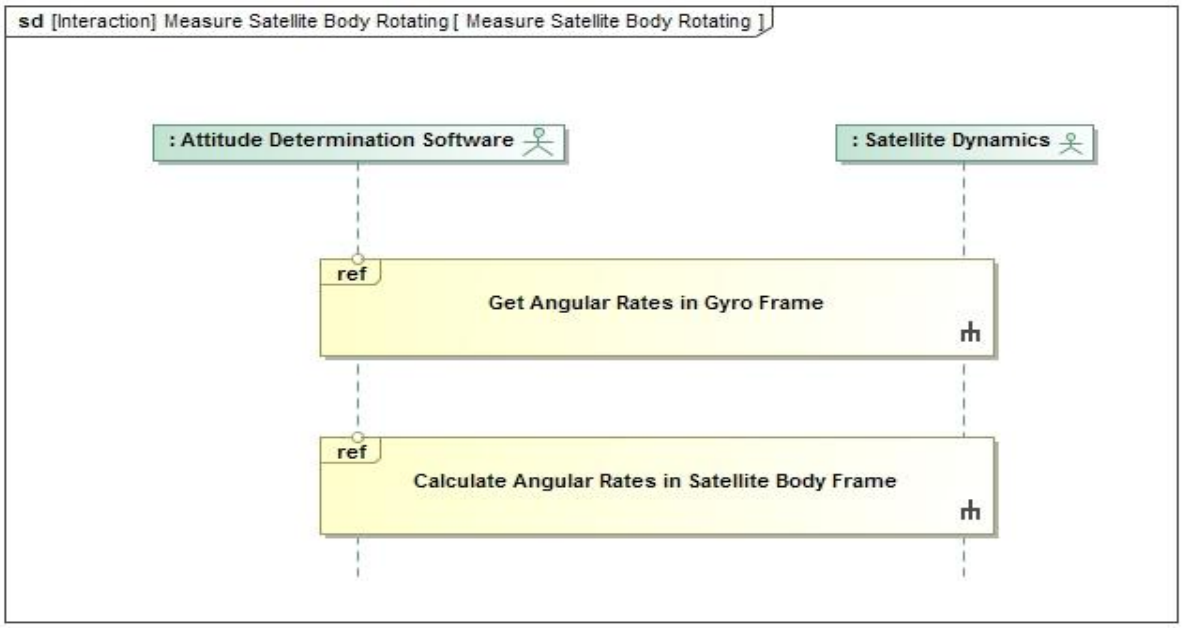

Figure III.1.10. The sequence diagram of “Measure Satellite Body Rotating”

After analyzing the functions of the attitude determination software, the software modules are developed.

The block definition diagram of "Attitude Determination Software"

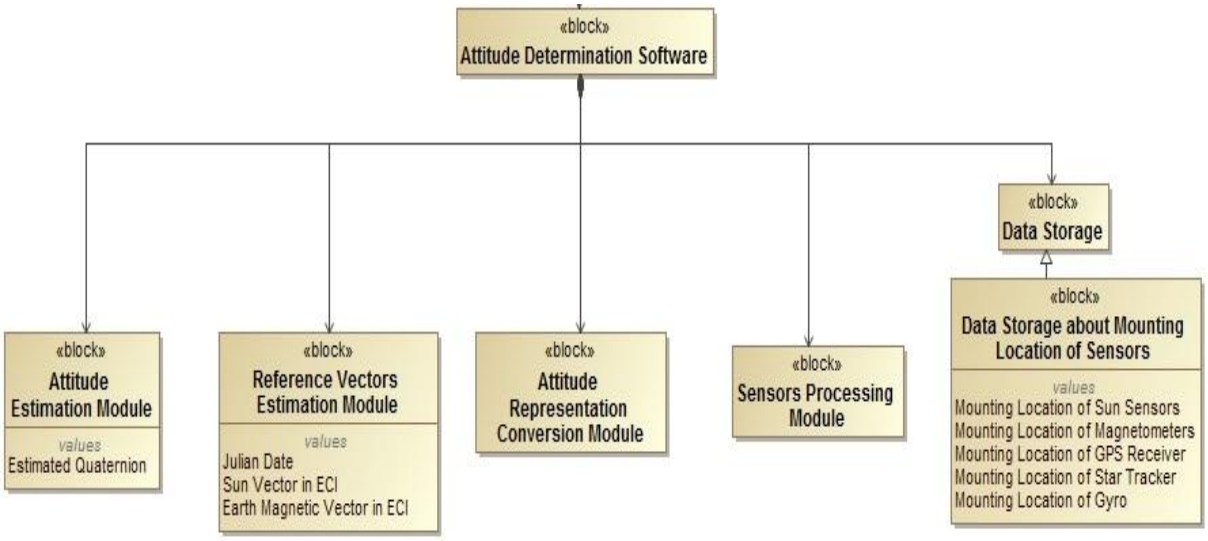

Figure III.1.11. The block definition diagram of "Attitude Determination Software"

The internal block diagram of "Attitude Determination Software"

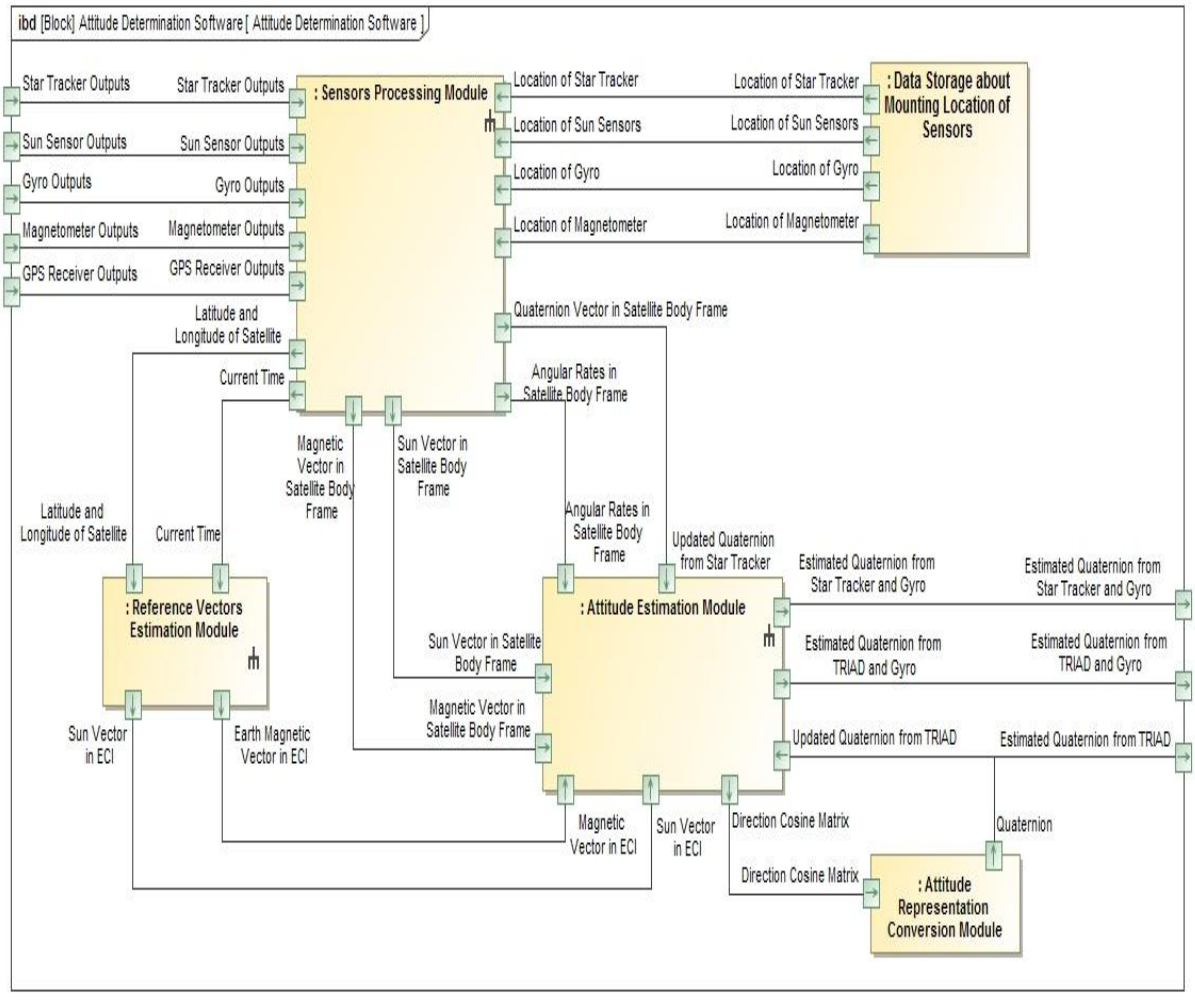

Figure III.1.12. The internal block diagram of "Attitude Determination Software"

The activity diagram of “Estimate Current Attitude”

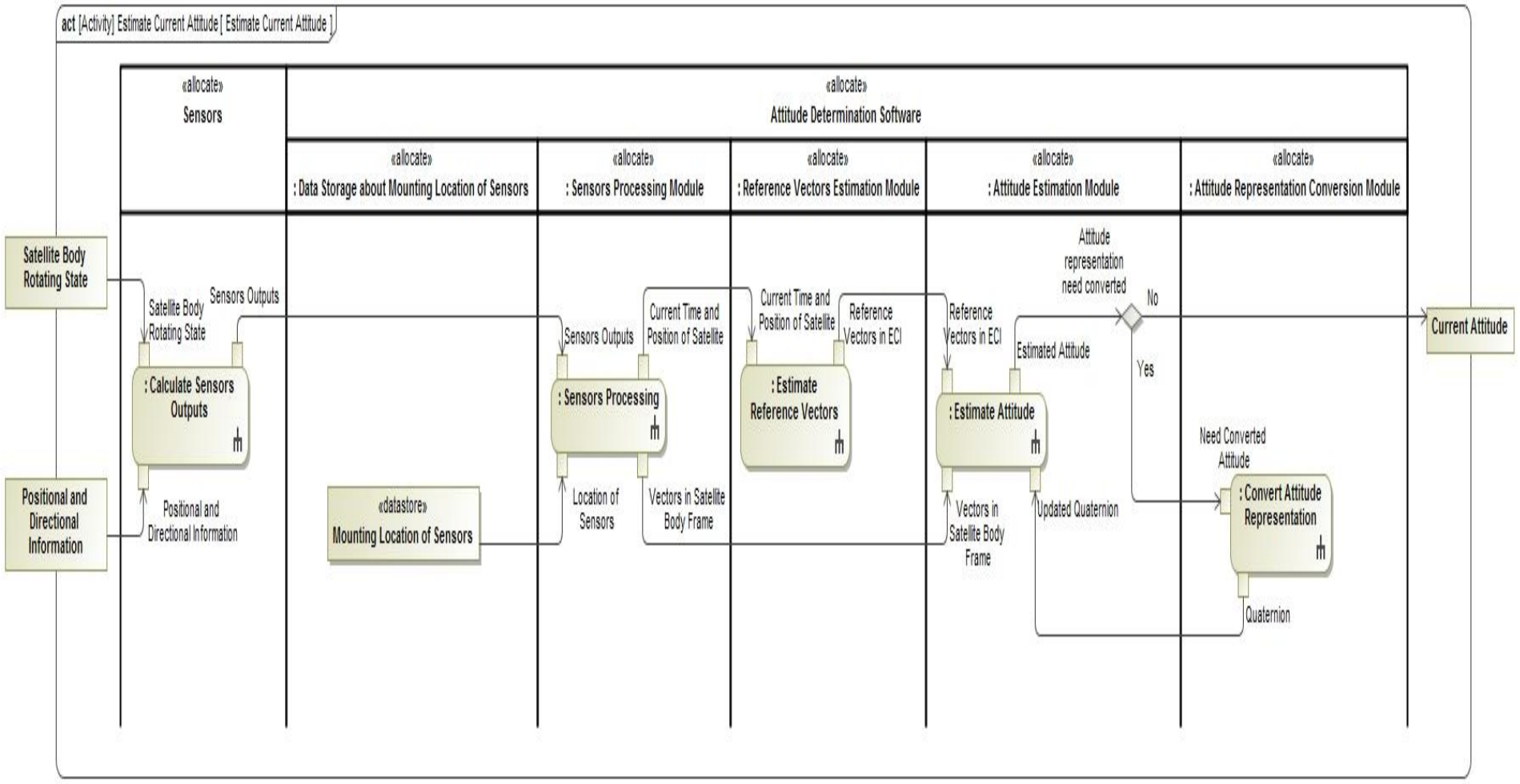

Figure III.1.13. The activity diagram of “Estimate Current Attitude”

The requirement diagram of “Attitude Determination Software”

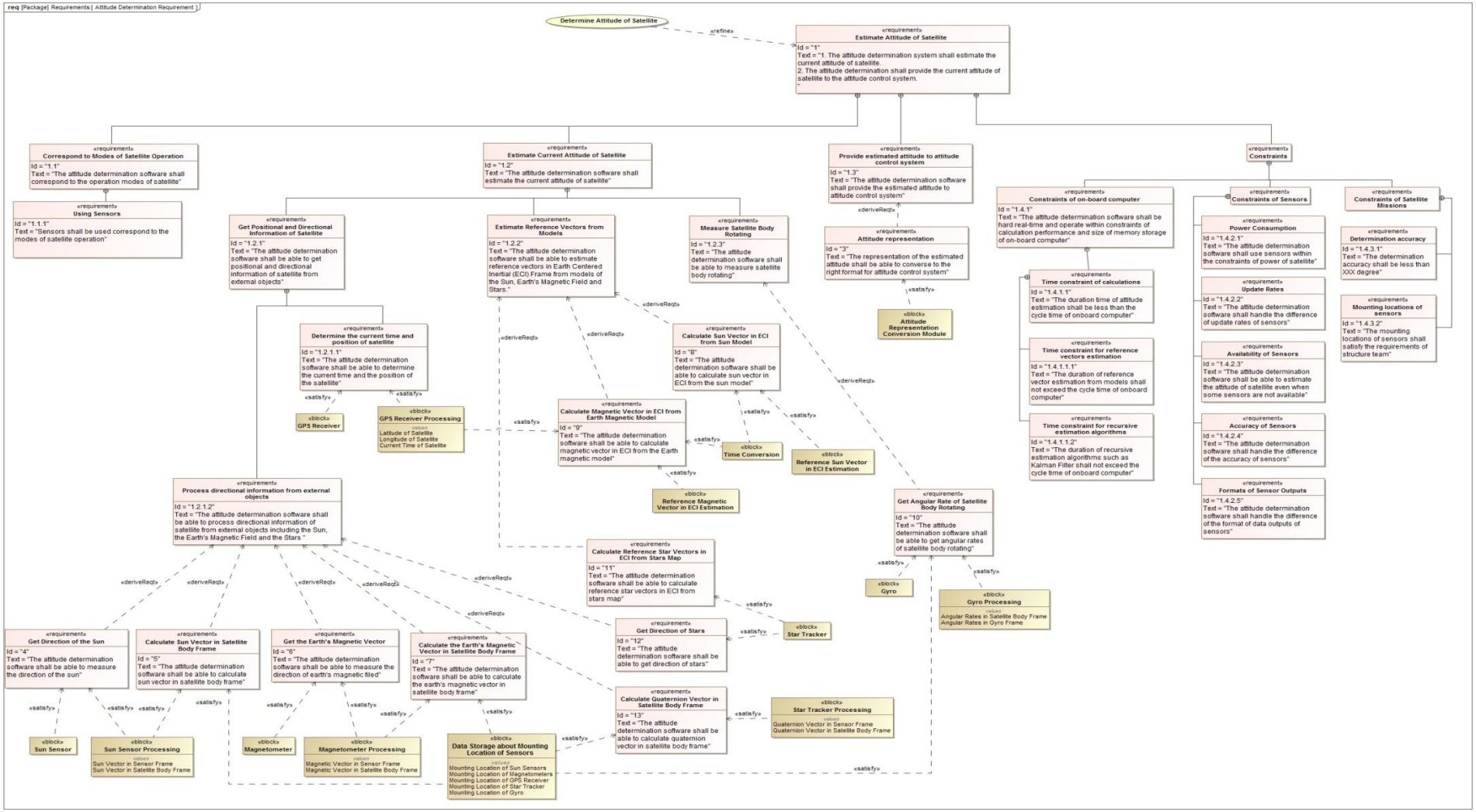

Figure III.1.14. The requirement diagram of “Attitude Determination Software”

III.2. Design of software modules

In this section, the design of “Sensors Processing Module”, “Reference Vectors Estimation Module” and “Attitude Estimation Module” are showed.

For each software module, the diagrams including the block definition diagram, the activity diagram and the internal block diagram are described.

The “Sensors Processing Module” consists of the software components to process the data outputs of each type of sensors including “Gyro Processing”, “Star Tracker Processing”, “GPS Receiver Processing”, “Magnetometer Processing” and “Sun Sensor Processing”.

The “Reference Vectors Estimation Module” contains the software components including “Time Conversion”, “Reference Sun Vector in ECI Estimation” and “Reference Magnetic Vector in ECI Estimation”.

The “Attitude Estimation Module” has three software components including “Attitude Estimation by TRIAD”, ““Attitude Estimation by TRIAD and Gyro” and “Attitude Estimation by Star Tracker and Gyro”.

For each software component, the consistency between sequence, activity and internal block diagrams are checked by analyzing the diagrams.

The block definition diagram of “Sensors Processing Module”

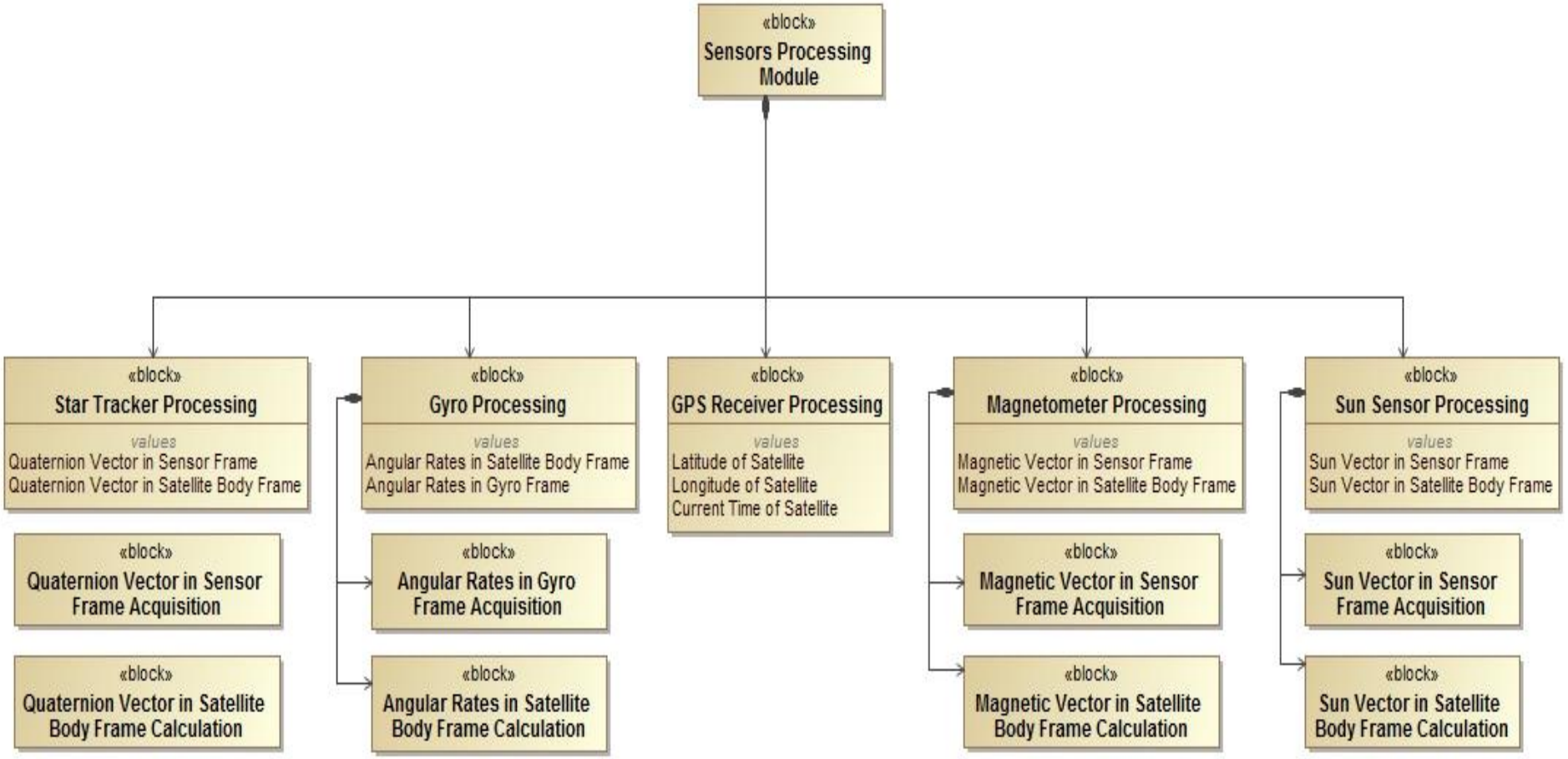

Figure III.2.1. The block definition diagram of “Sensors Processing Module”

The activity diagram of “Sensors Processing Module”

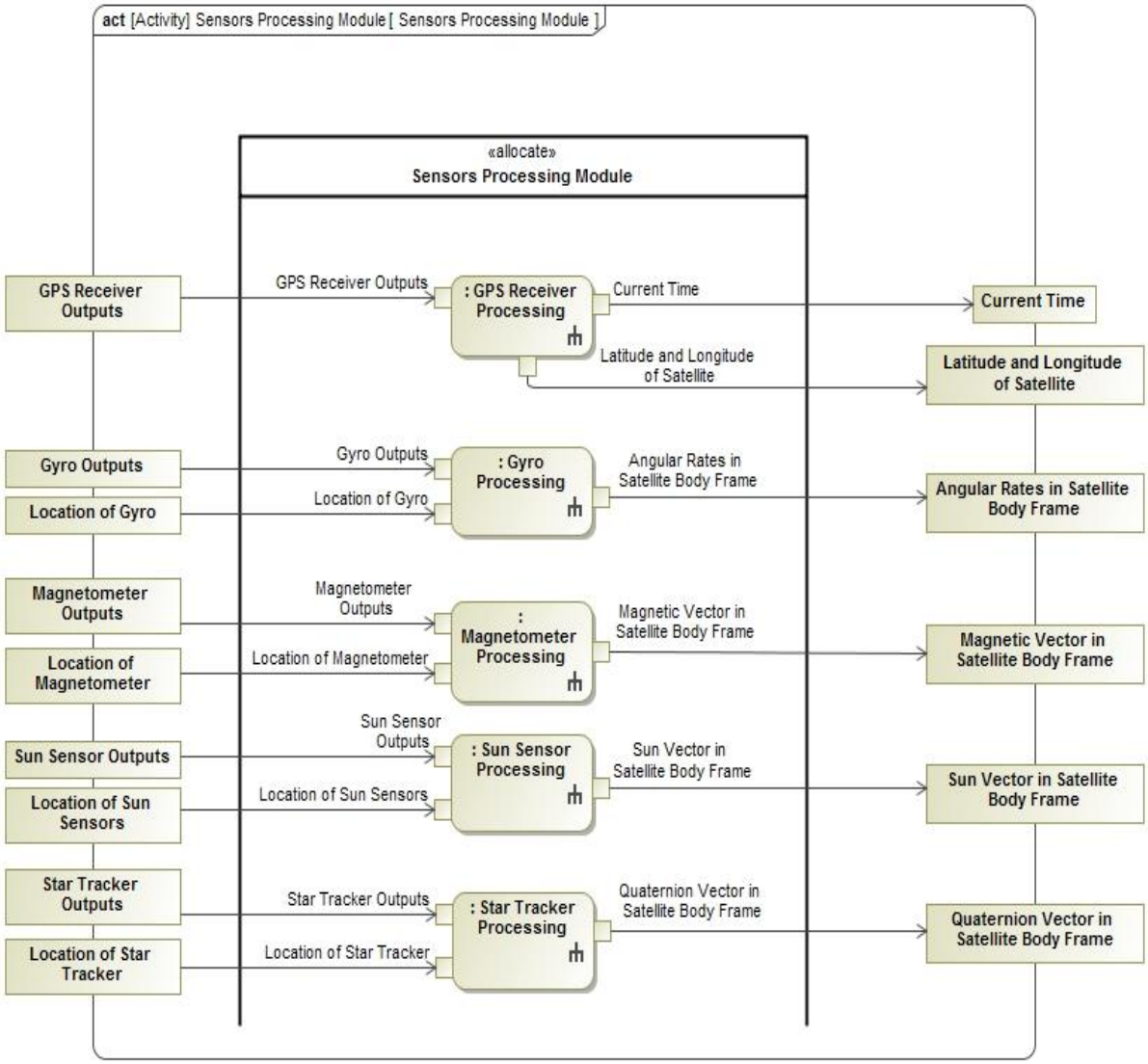

Figure III.2.2. The activity diagram of “Sensors Processing Module”

The internal block diagram of “Sensors Processing Module”

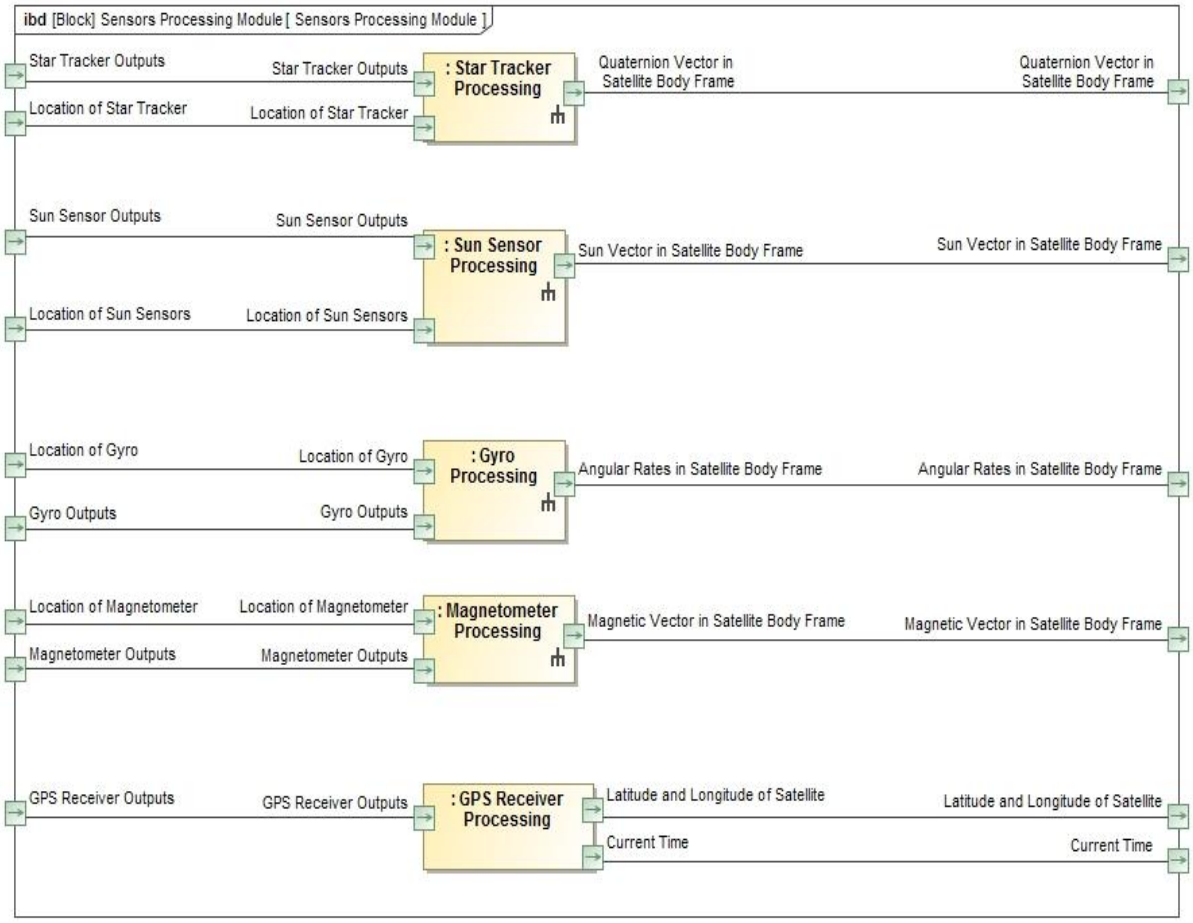

Figure III.2.3. The internal block diagram of “Sensors Processing Module”

The block definition diagram of “Reference Vectors Estimation Module”

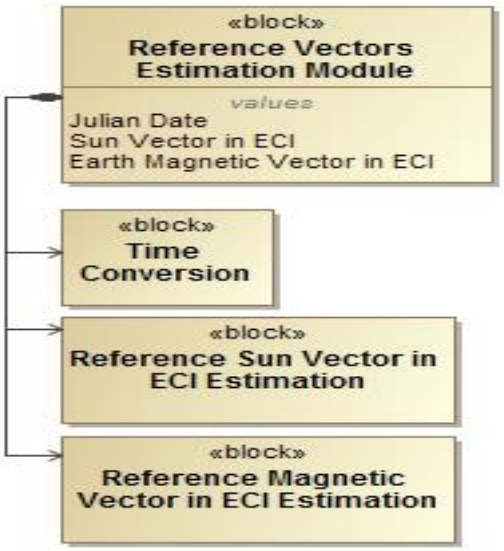

Figure III.2.4. The block definition diagram of “Reference Vectors Estimation Module”

The activity diagram of “Reference Vectors Estimation Module”

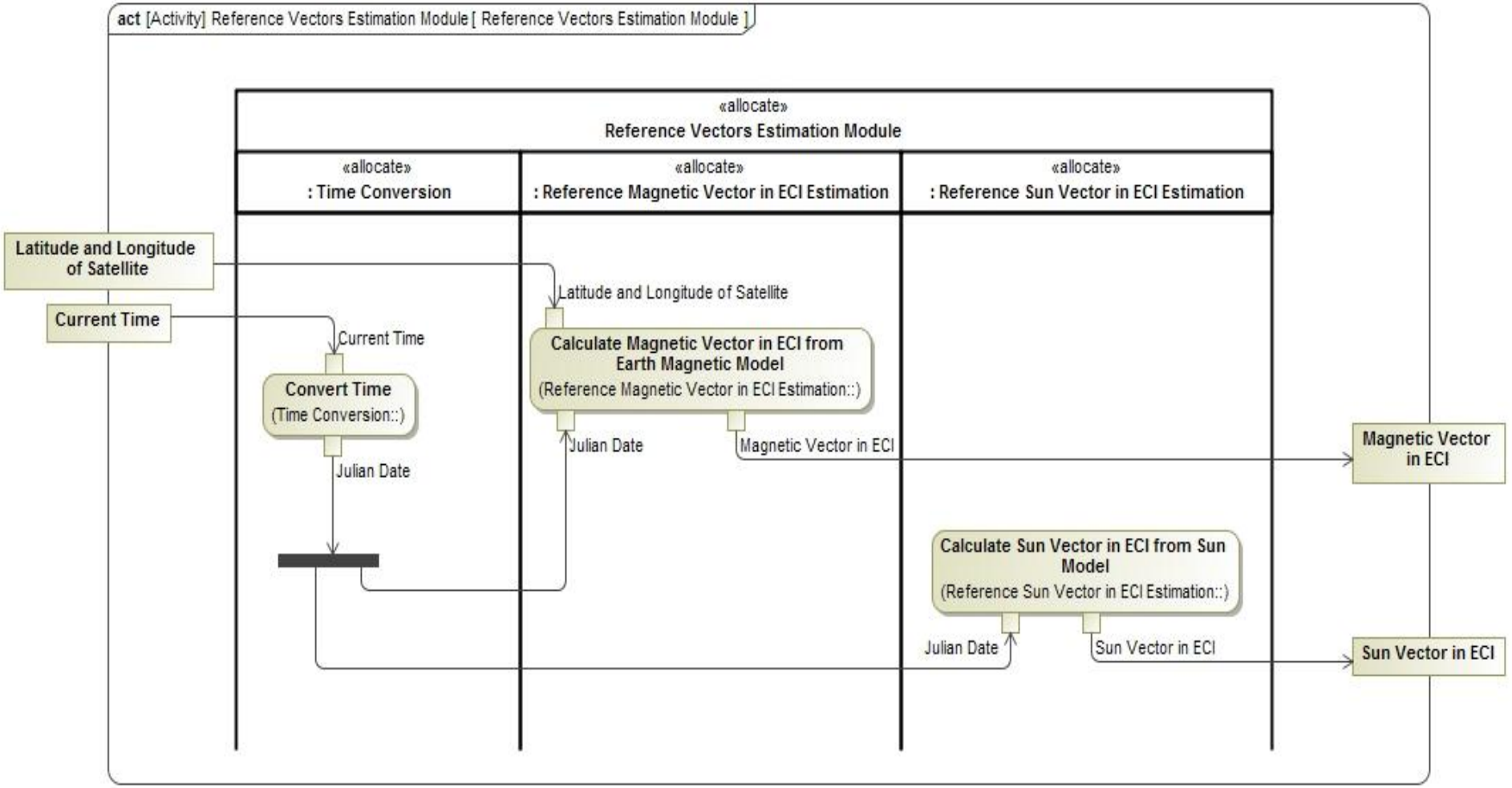

Figure III.2.5. The activity diagram of “Reference Vectors Estimation Module”

The internal block diagram of "Reference Vectors Estimation Module"

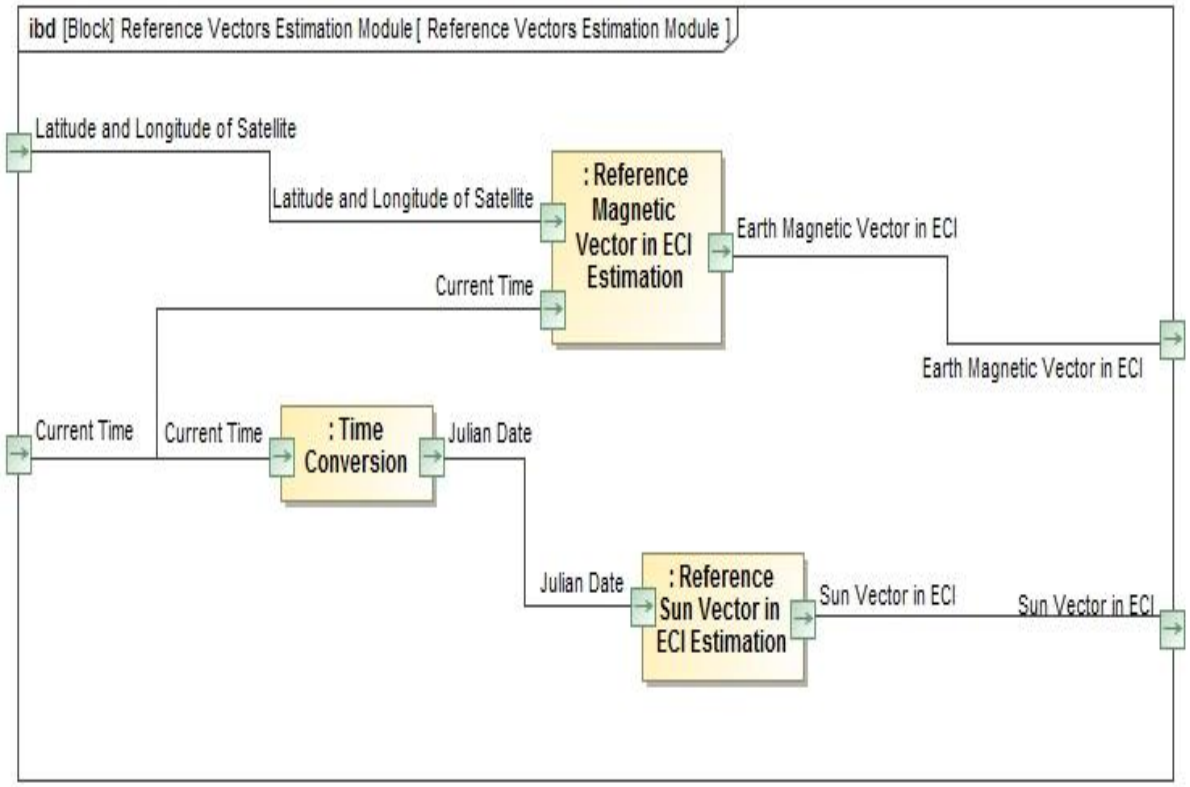

Figure III.2.6. The internal block diagram of "Reference Vectors Estimation Module"

The implementation in C code for the software component "Time Conversion" and "Reference Sun Vector in ECI Estimation" are showed in the Appendix.

The block definition diagram of "Attitude Estimation Module"

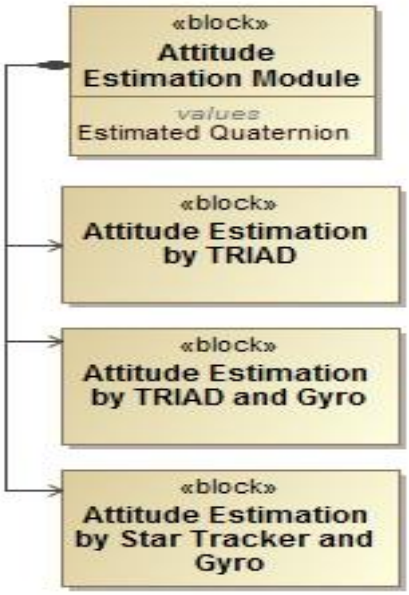

Figure III.2.7. The block definition diagram of "Attitude Estimation Module"

The activity diagram of “Attitude Estimation Module”

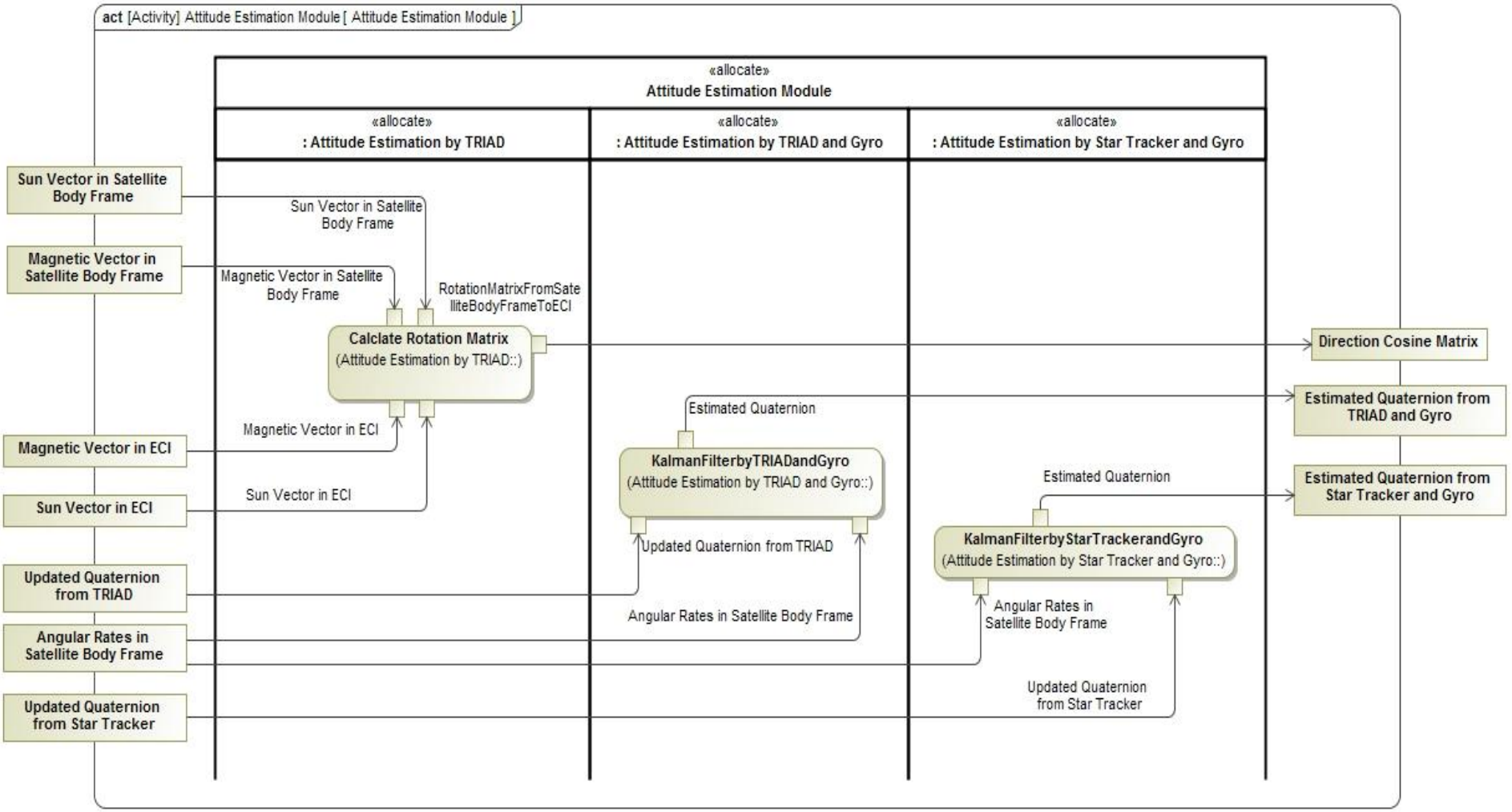

Figure III.2.8. The activity diagram of “Attitude Estimation Module”

The internal block diagram of "Attitude Estimation Module"

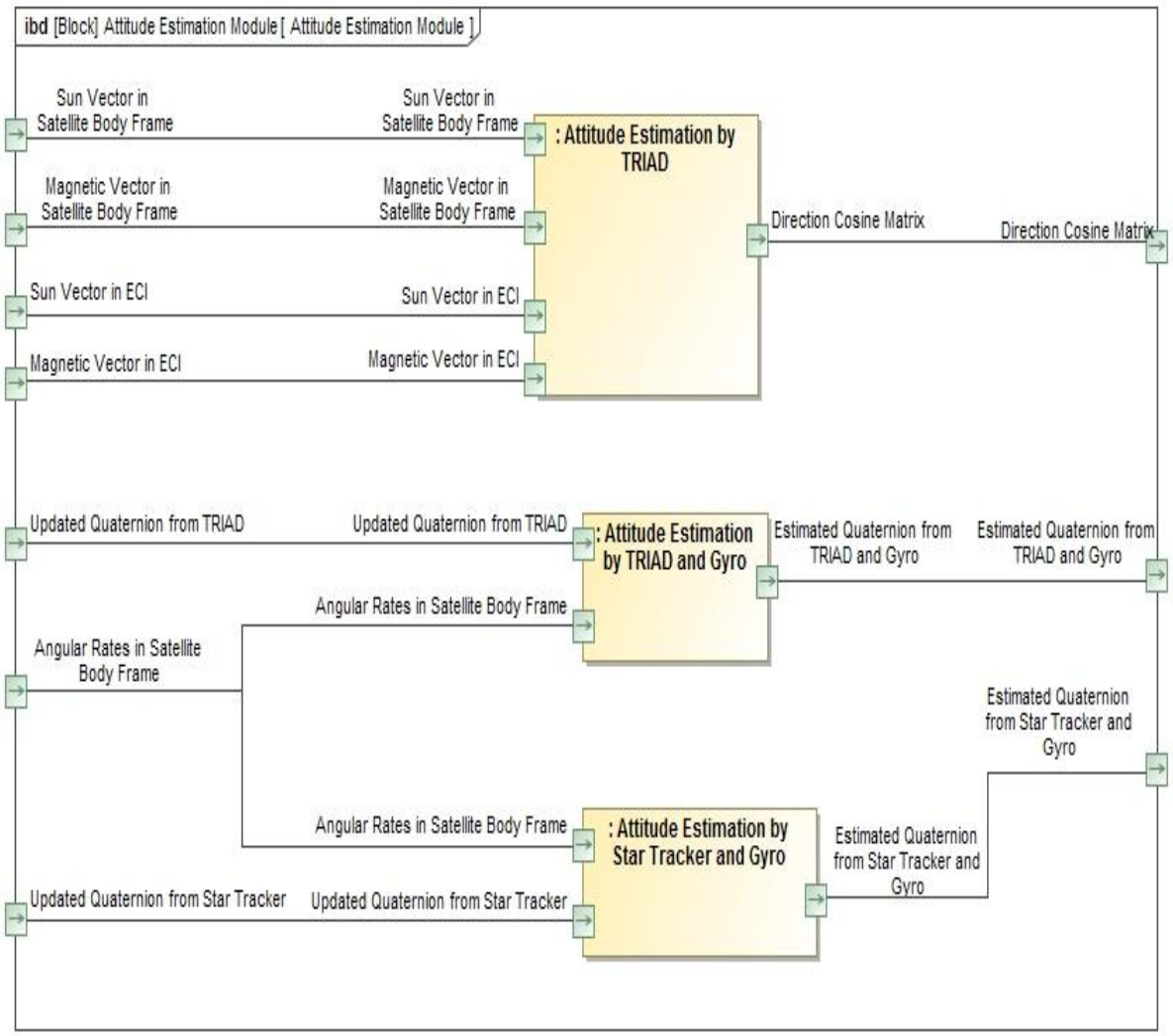

Figure III.2.9. The internal block diagram of "Attitude Estimation Module"

The implementation in C code for the software component "Attitude Estimation by TRIAD" is showed in the Appendix.

The sequence diagram of “Estimate Current Attitude of Satellite”

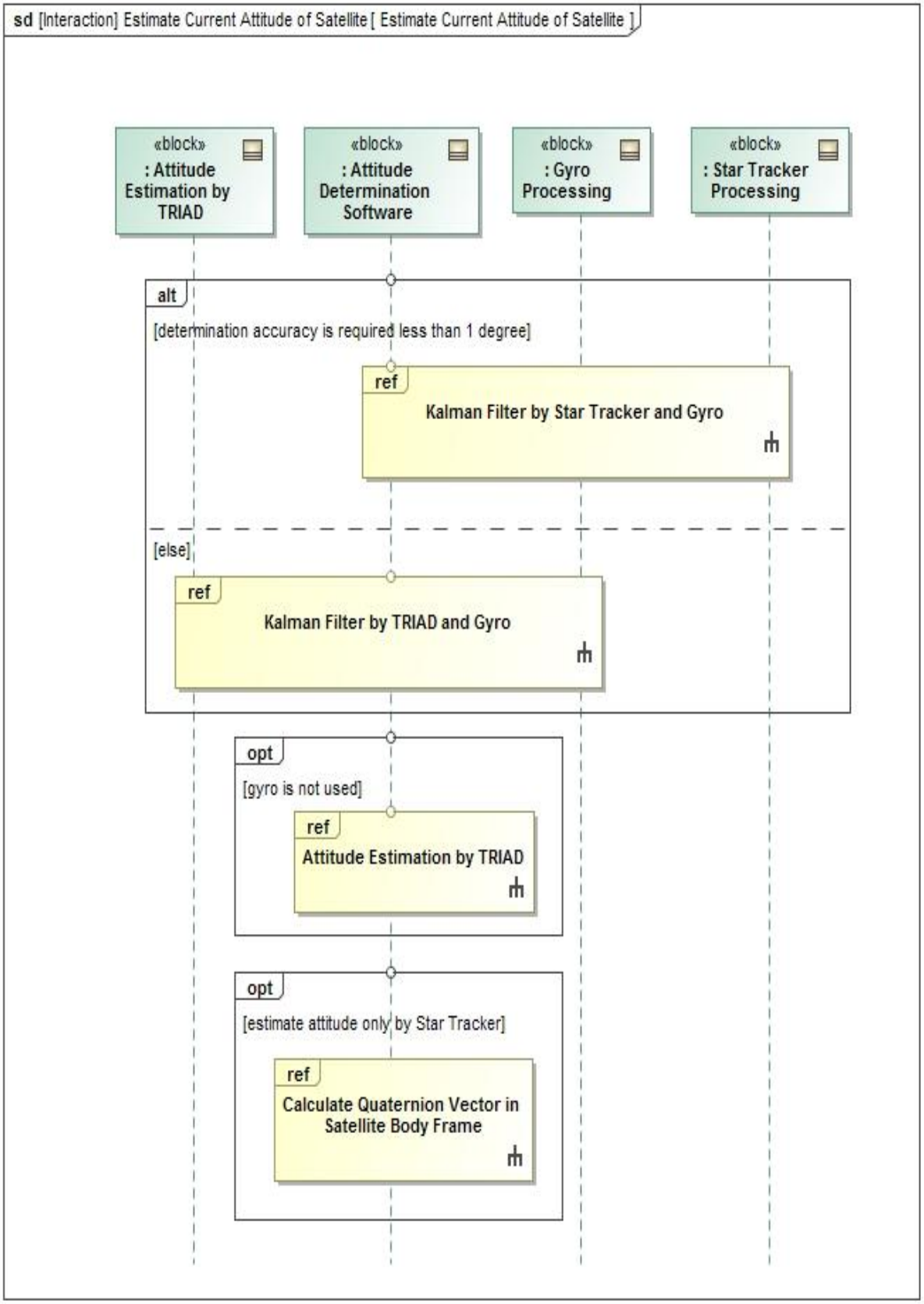

Figure III.2.10. The sequence diagram of “Estimate Current Attitude of Satellite”

III.3. Design of software components of sensors processing module

III.3.1. GPS Receiver Processing

The sequence diagrams of "GPS Receiver Processing" software component

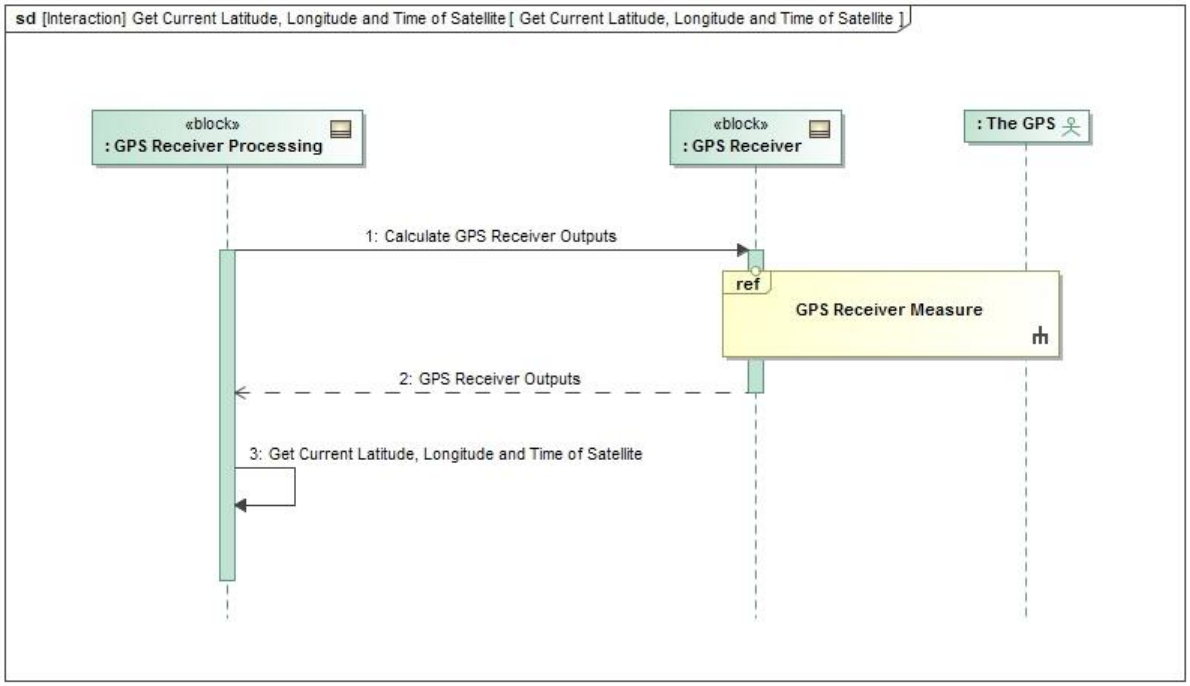

Figure III.3.1.1. The sequence diagram "Get Current Latitude, Longitude and Time of Satellite"

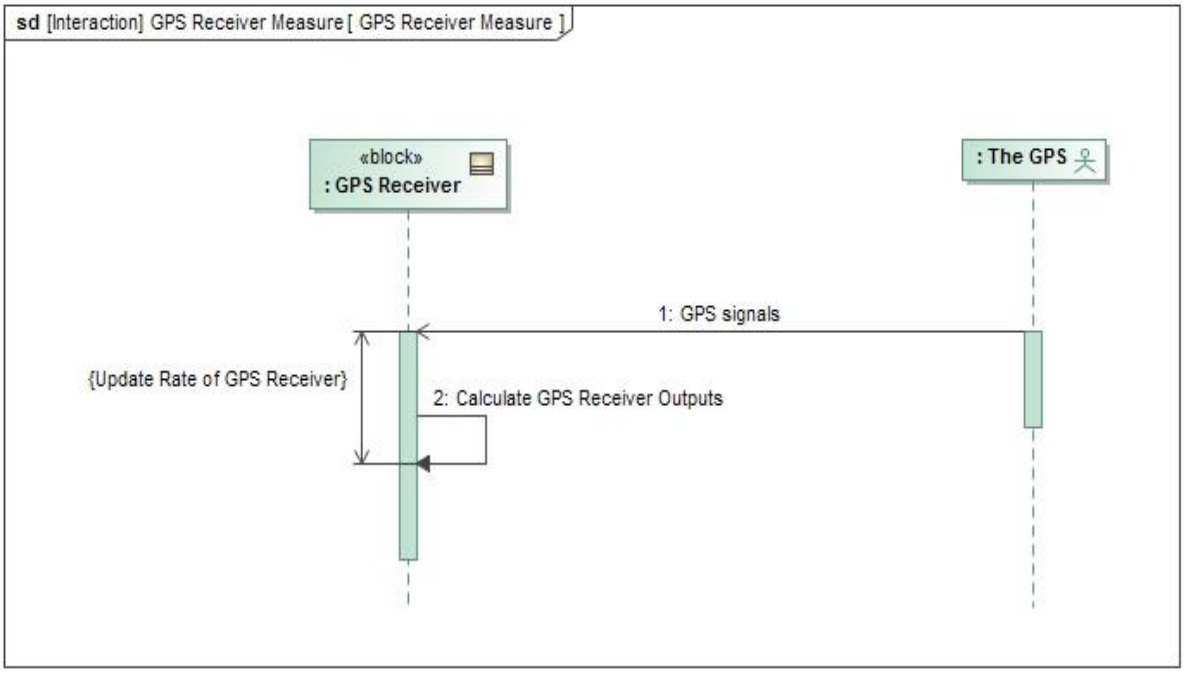

Figure III.3.1.2. The sequence diagram "GPS Receiver Measure"

The activity diagram of “GPS Receiver Processing”

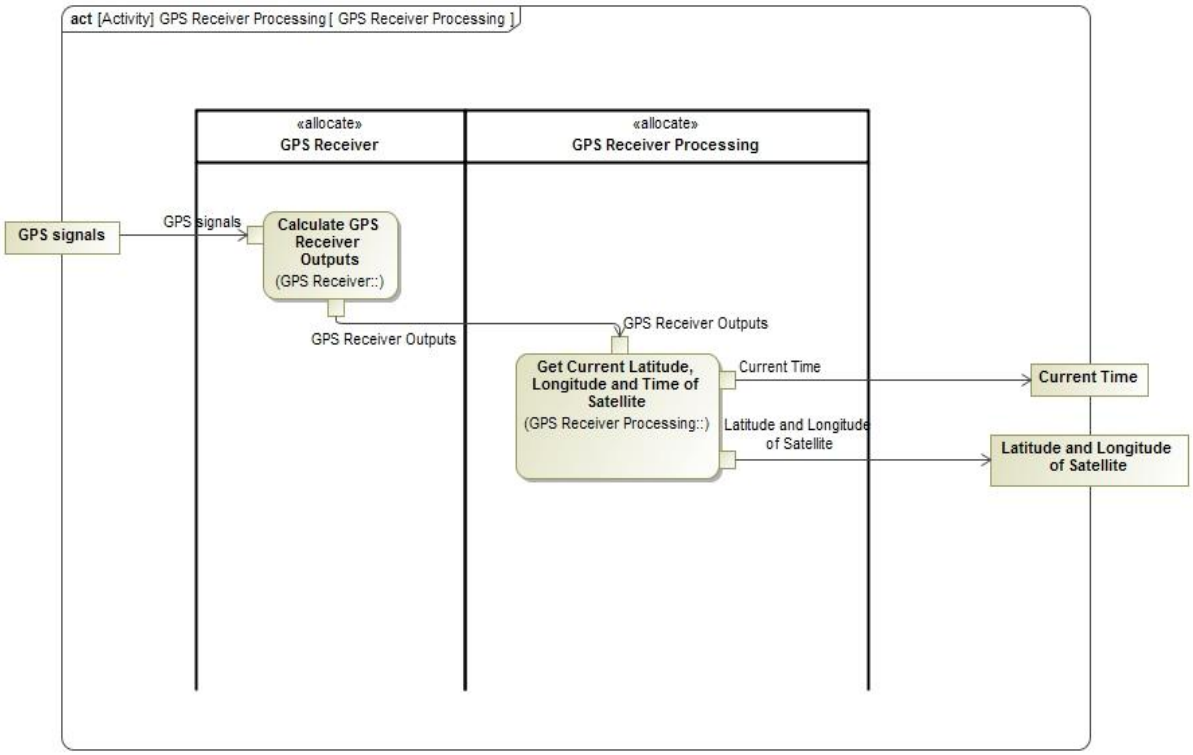

Figure III.3.1.3. The activity diagram of “GPS Receiver Processing”

III.3.2. Gyro Processing

The sequence diagrams of “Gyro Processing”

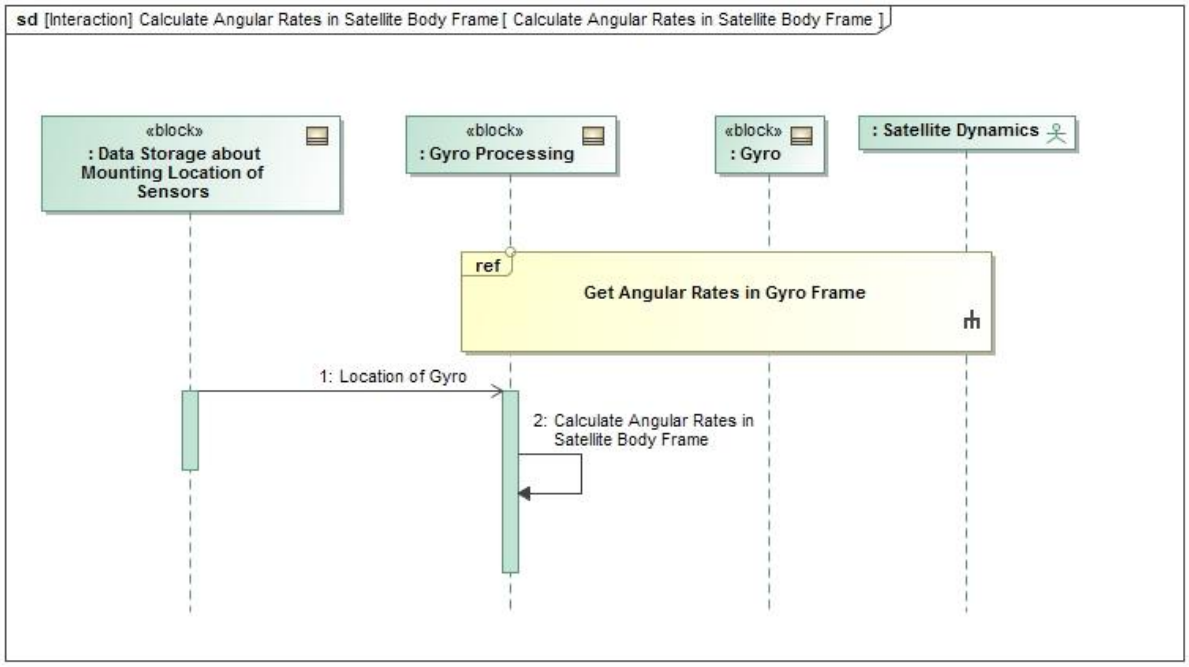

Figure III.3.2.1. The sequence diagram “Calculate Angular Rates in Satellite Body Frame”

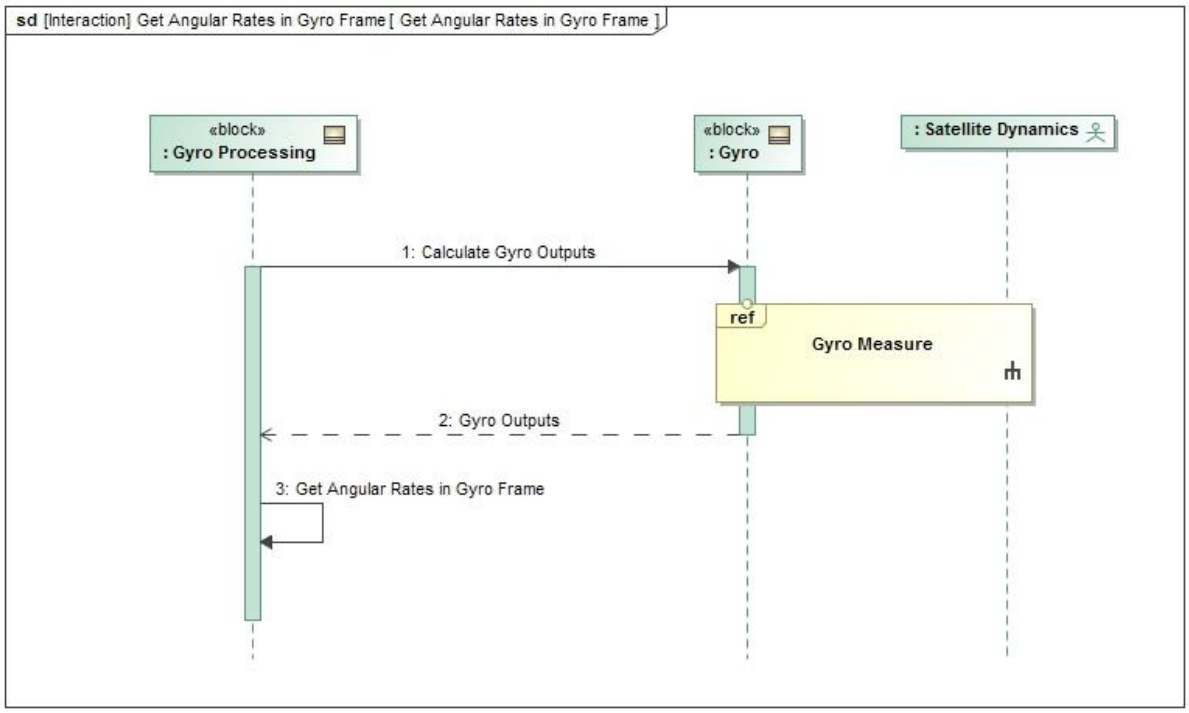

Figure III.3.2.2. The sequence diagram “Get Angular Rates in Gyro Frame”

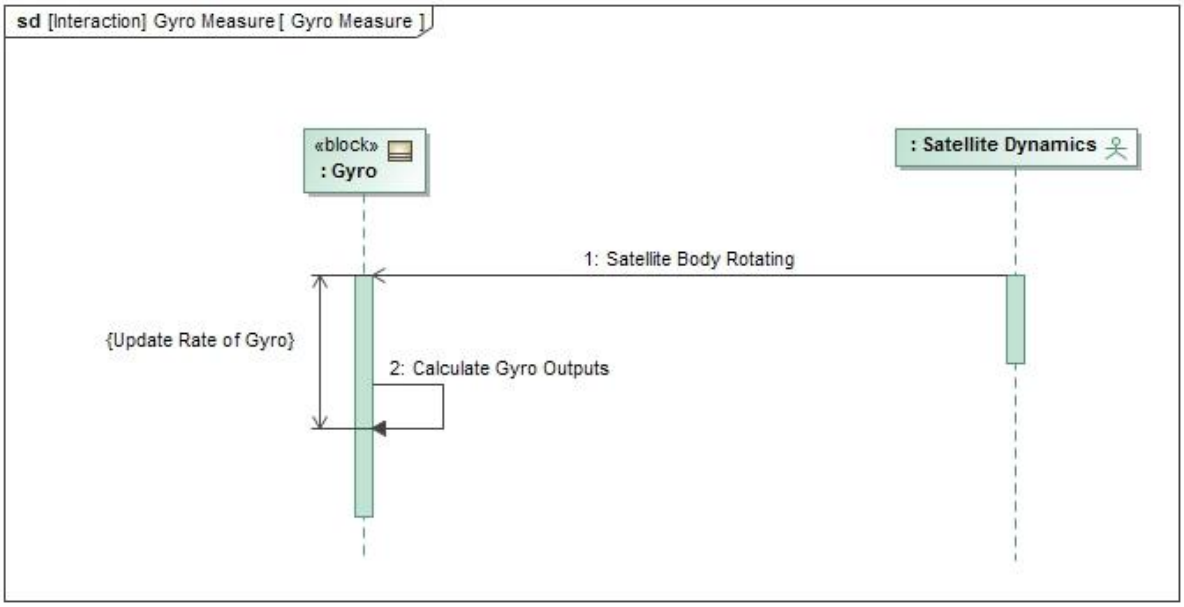

Figure III.3.2.3. The sequence diagram “Gyro Measure”

The activity diagram of “Gyro Processing”

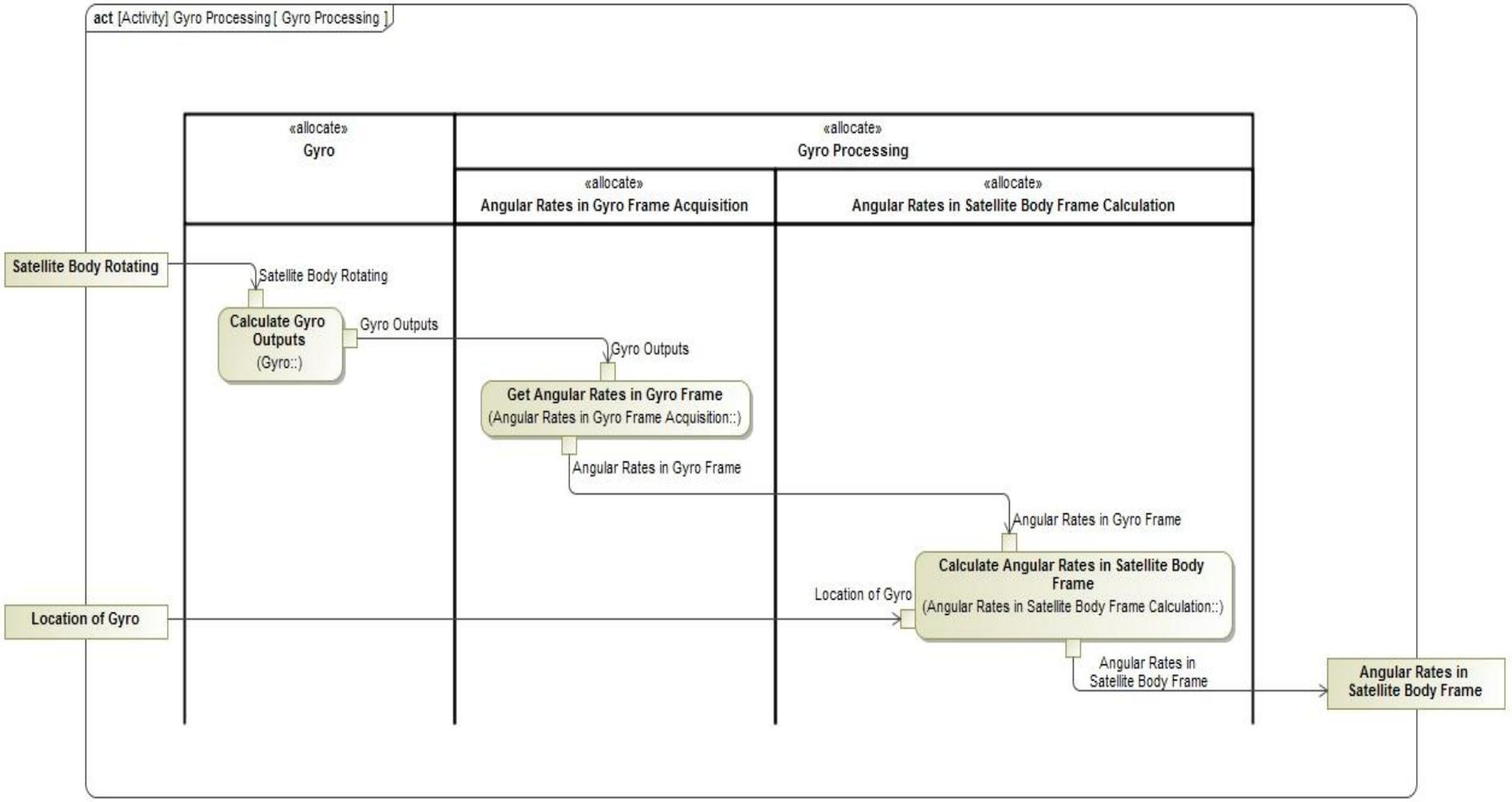

Figure III.3.2.4. The activity diagram of “Gyro Processing”

The internal block diagram of “Gyro Processing”

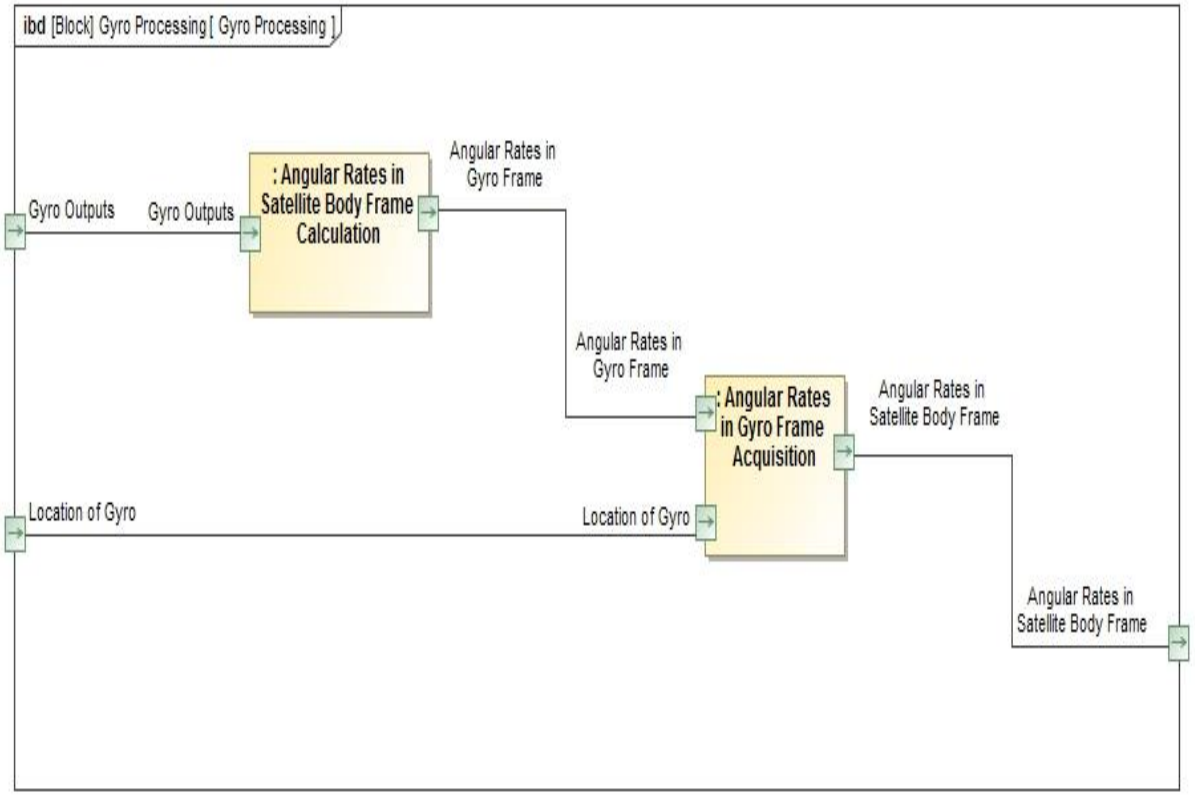

Figure III.3.2.5. The internal block diagram “Gyro Processing”

III.3.3. Magnetometer Processing

The sequence diagrams of “Magnetometer Processing”

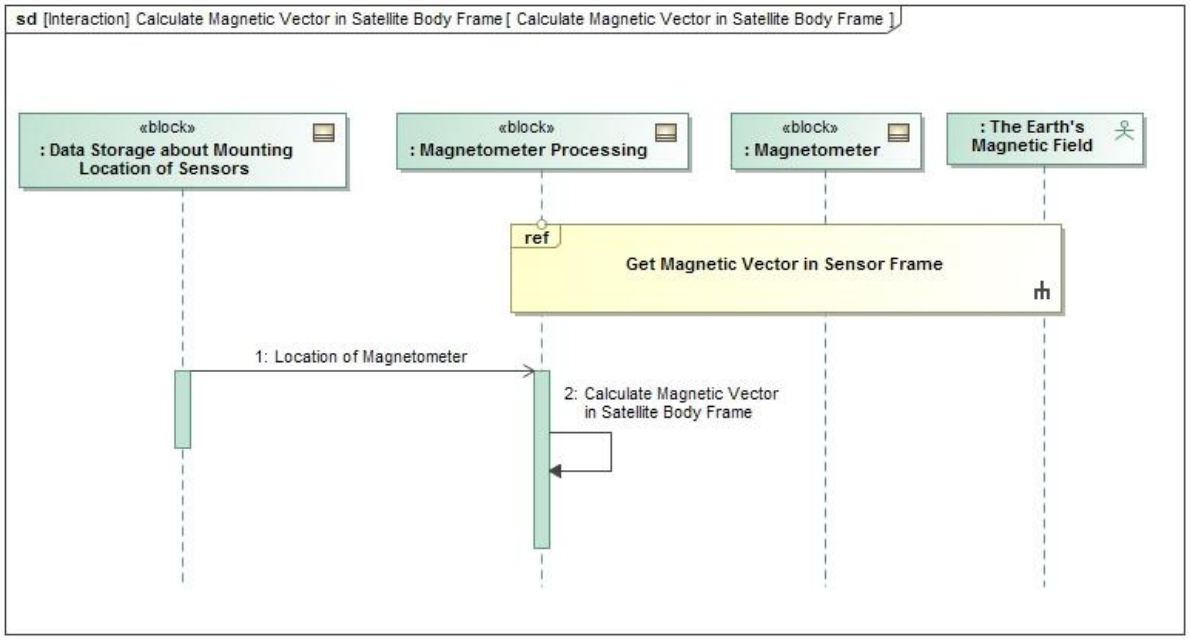

Figure III.3.3.1. The sequence diagram “Calculate Magnetic Vector in Satellite Body Frame”

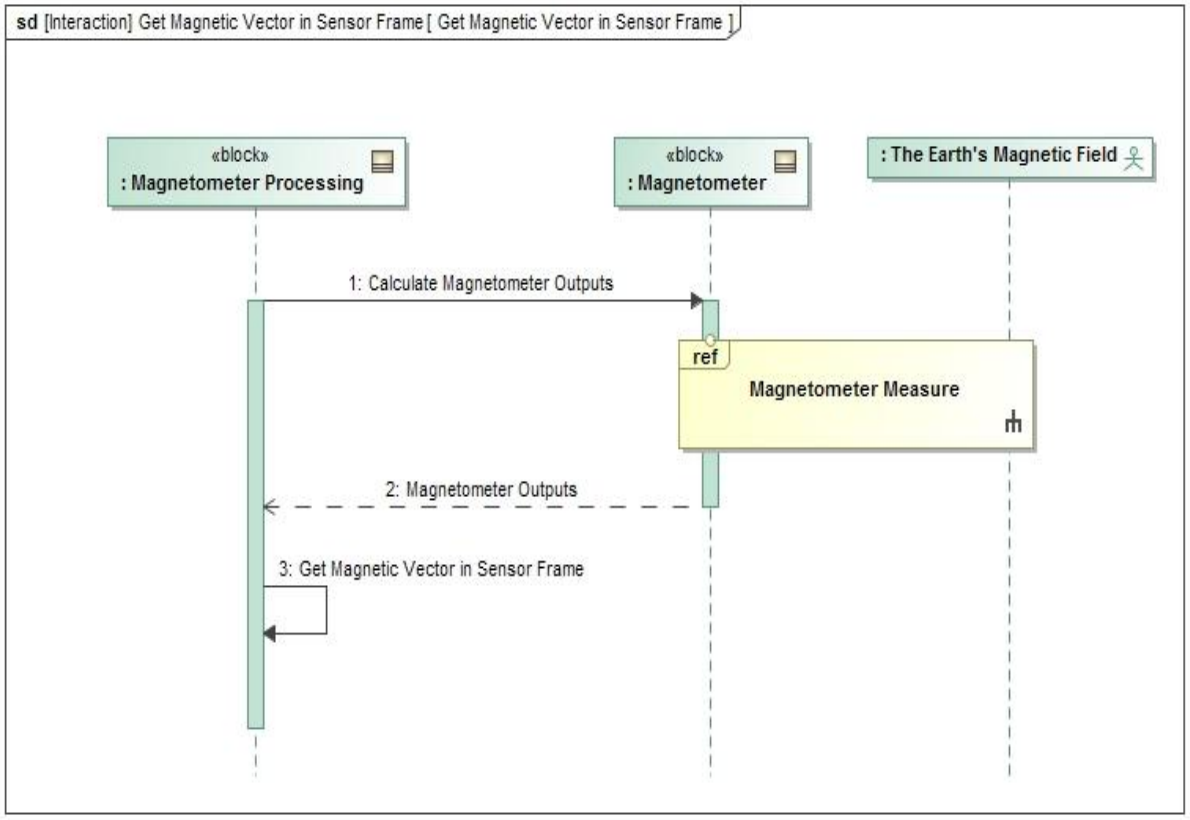

Figure III.3.3.2. The sequence diagram “Get Magnetic Vector in Sensor Frame”

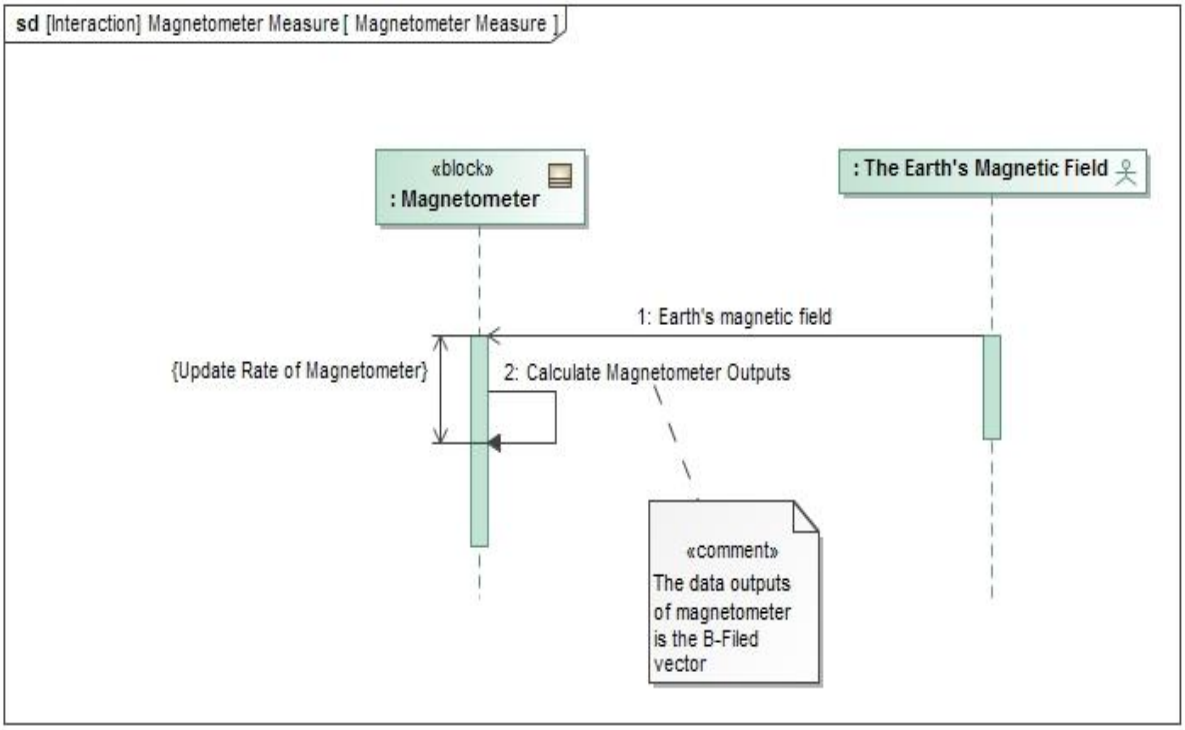

Figure III.3.3.3. The sequence diagram “Magnetometer Measure”

The activity diagram of “Magnetometer Processing”

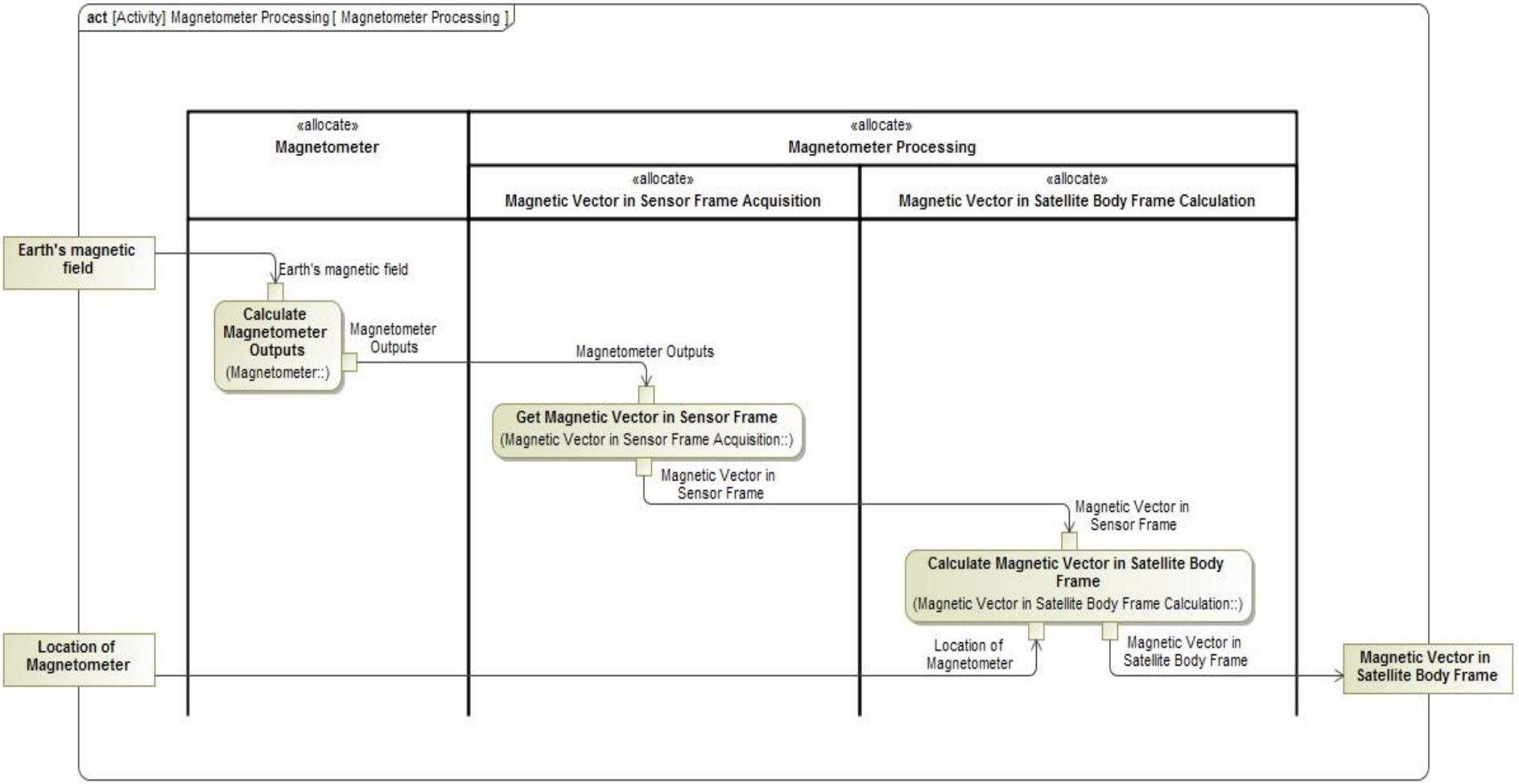

Figure III.3.3.4. The activity diagram of “Magnetometer Processing”

The internal block diagram of “Magnetometer Processing”

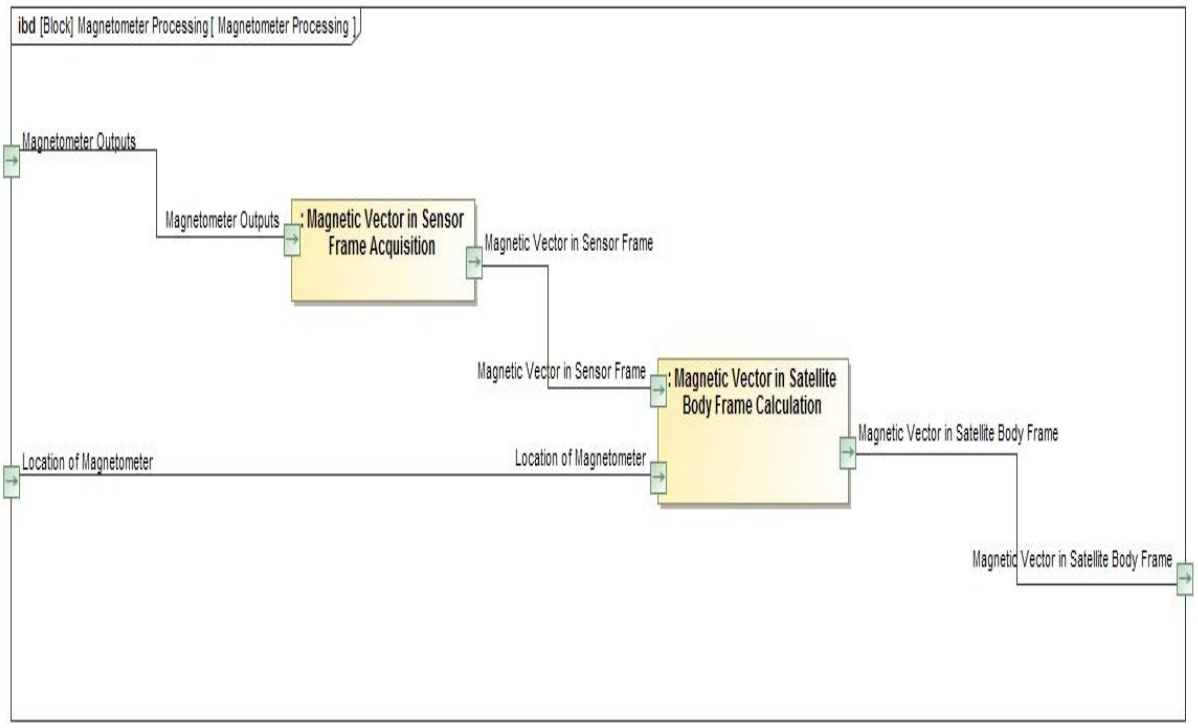

Figure III.3.3.5. The internal block diagram of “Magnetometer Processing”

III.3.4. Star Tracker Processing

The sequence diagrams of “Star Tracker Processing”

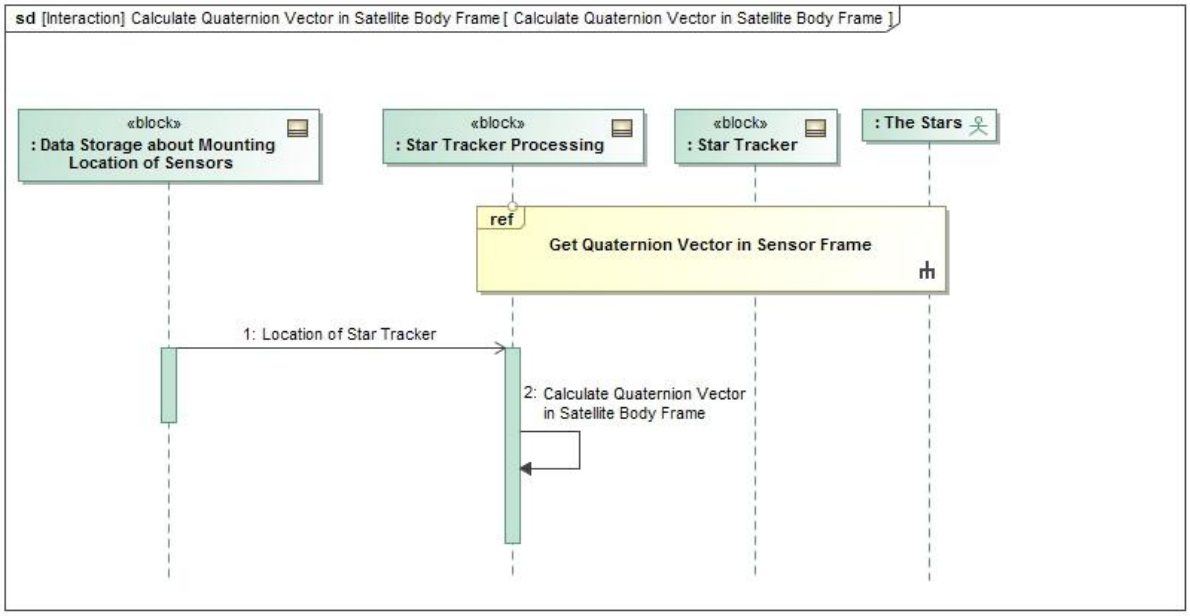

Figure III.3.4.1. The sequence diagram “Calculate Quaternion Vector in Satellite Body Frame”

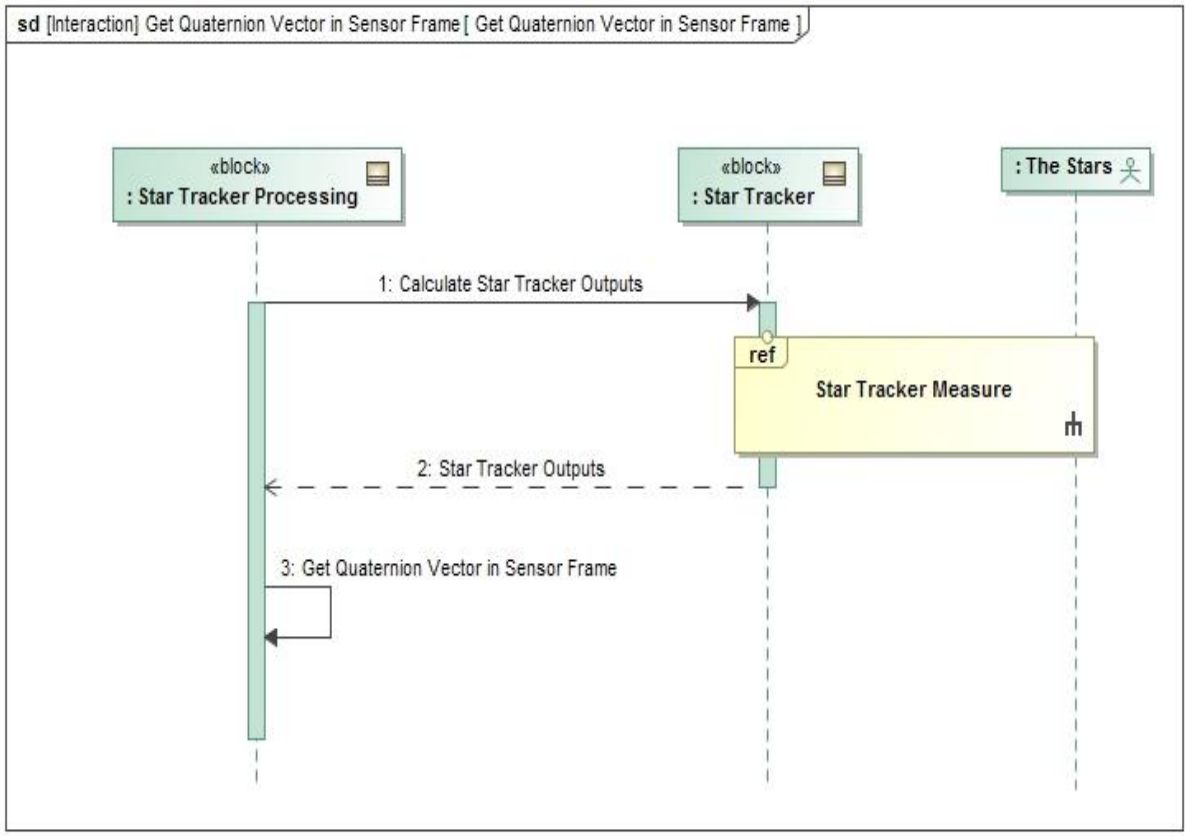

Figure III.3.4.2. The sequence diagram “Get Quaternion Vector in Sensor Frame”

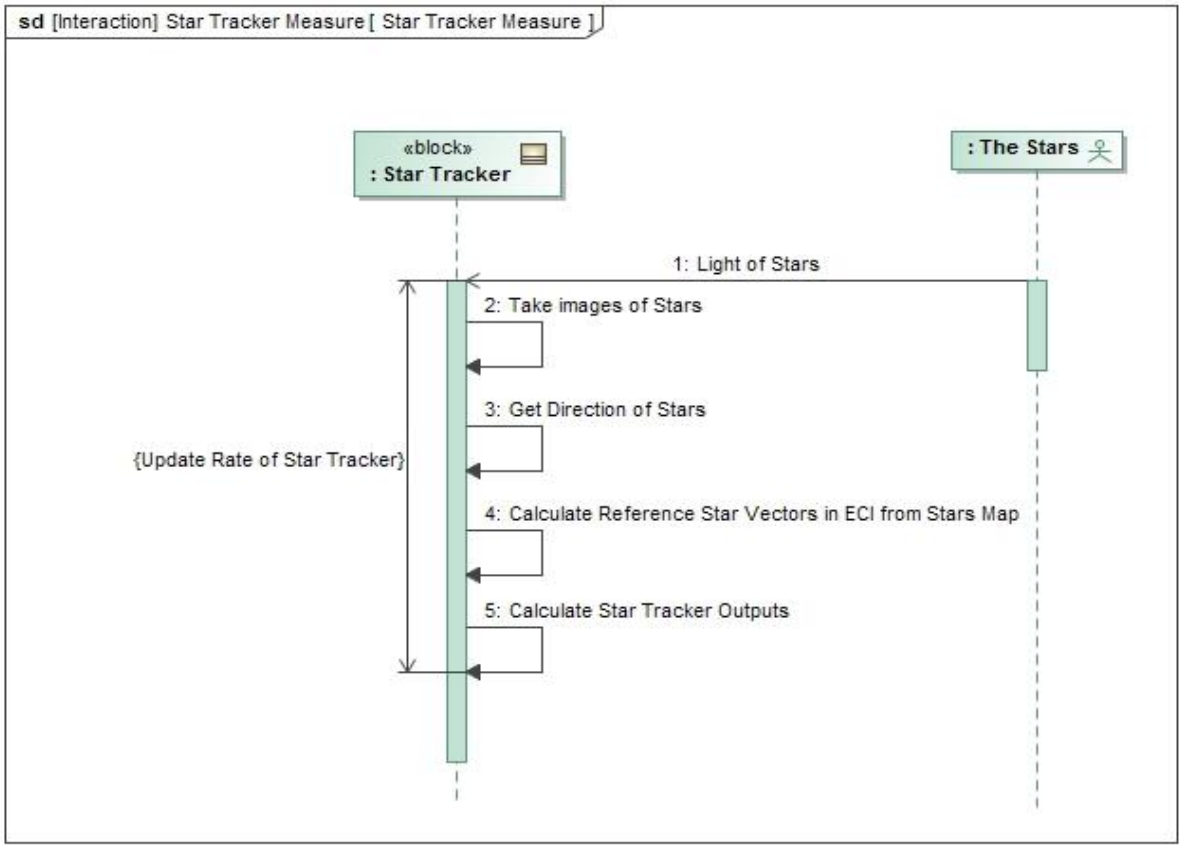

Figure III.3.4.3. The sequence diagram “Star Tracker Measure”

The activity of "Star Tracker Processing"

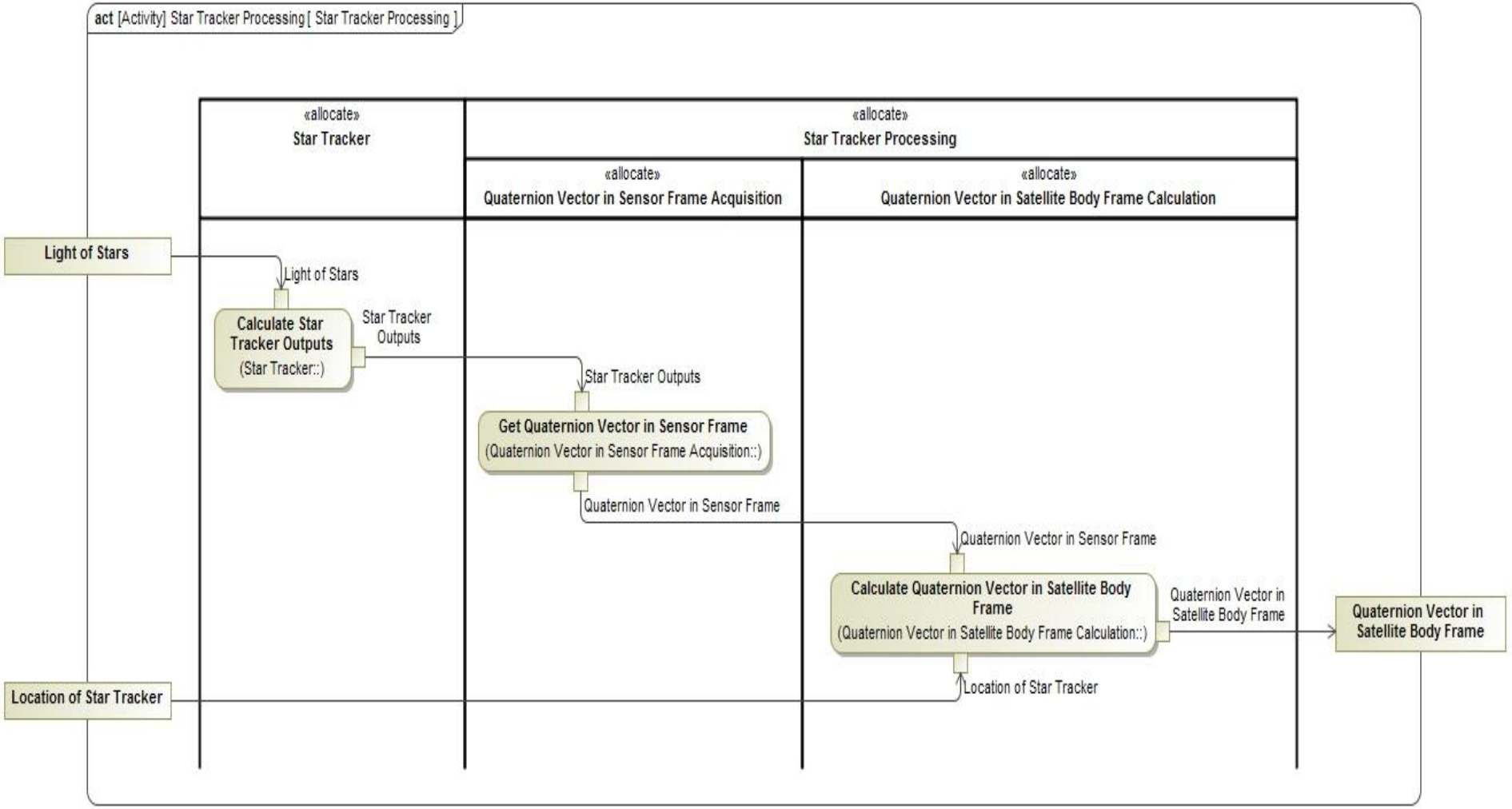

Figure III.3.4.4. The activity of "Star Tracker Processing"

The internal block diagram of “Star Tracker Processing”

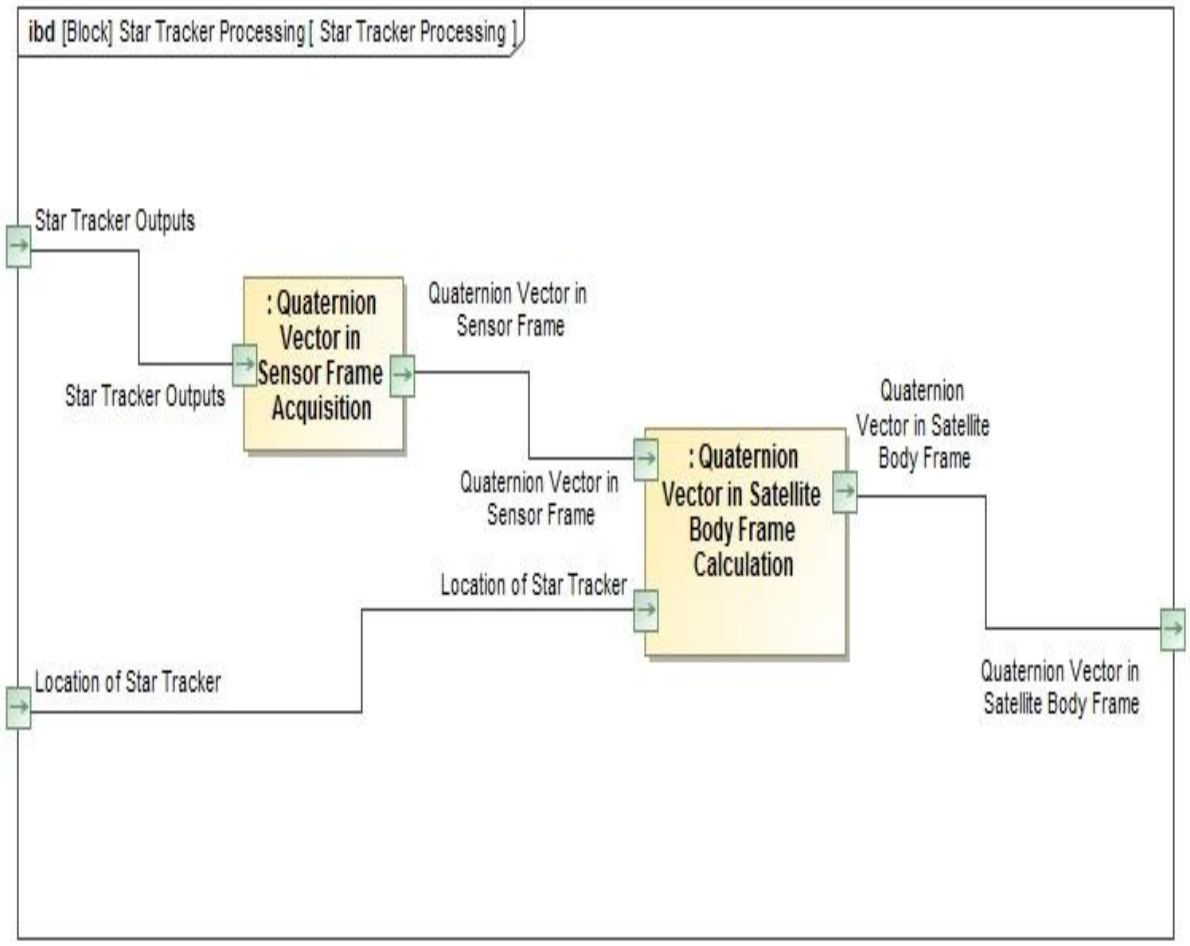

Figure III.3.4.5. The internal block diagram of “Star Tracker Processing”

III.3.5. Sun Sensor Processing

The sequence diagrams of “Sun Sensor Processing”

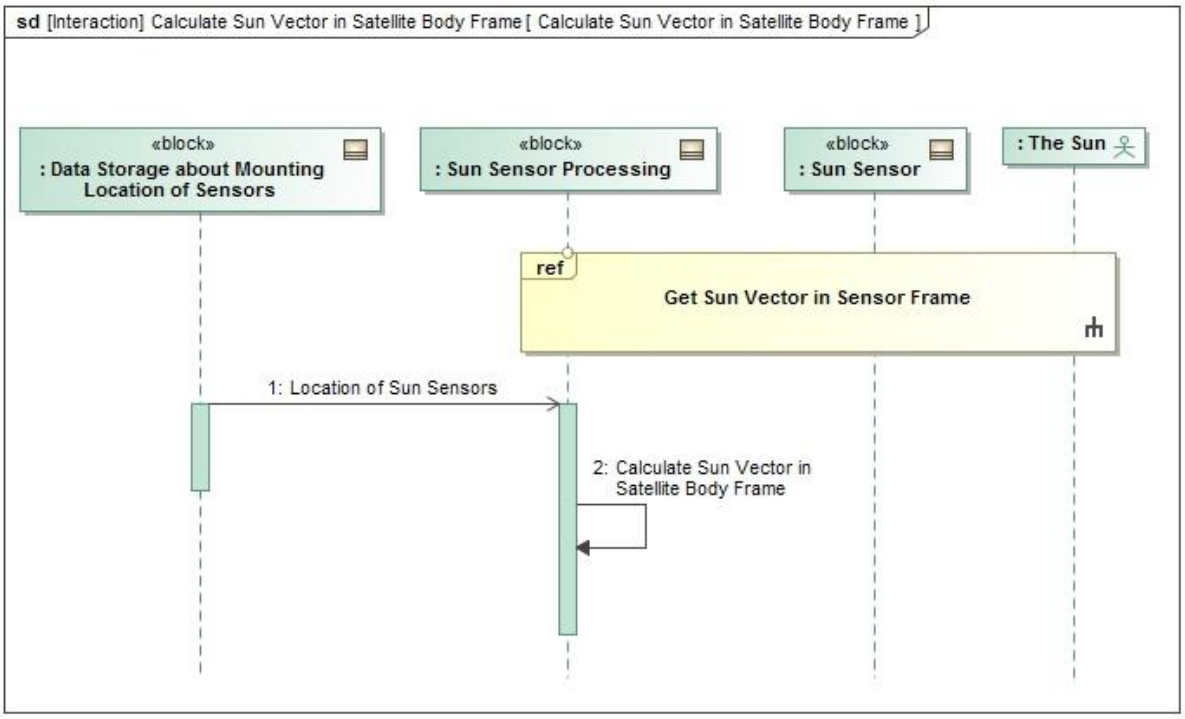

Figure III.3.5.1. The sequence diagram “Calculate Sun Vector in Satellite Body Frame”

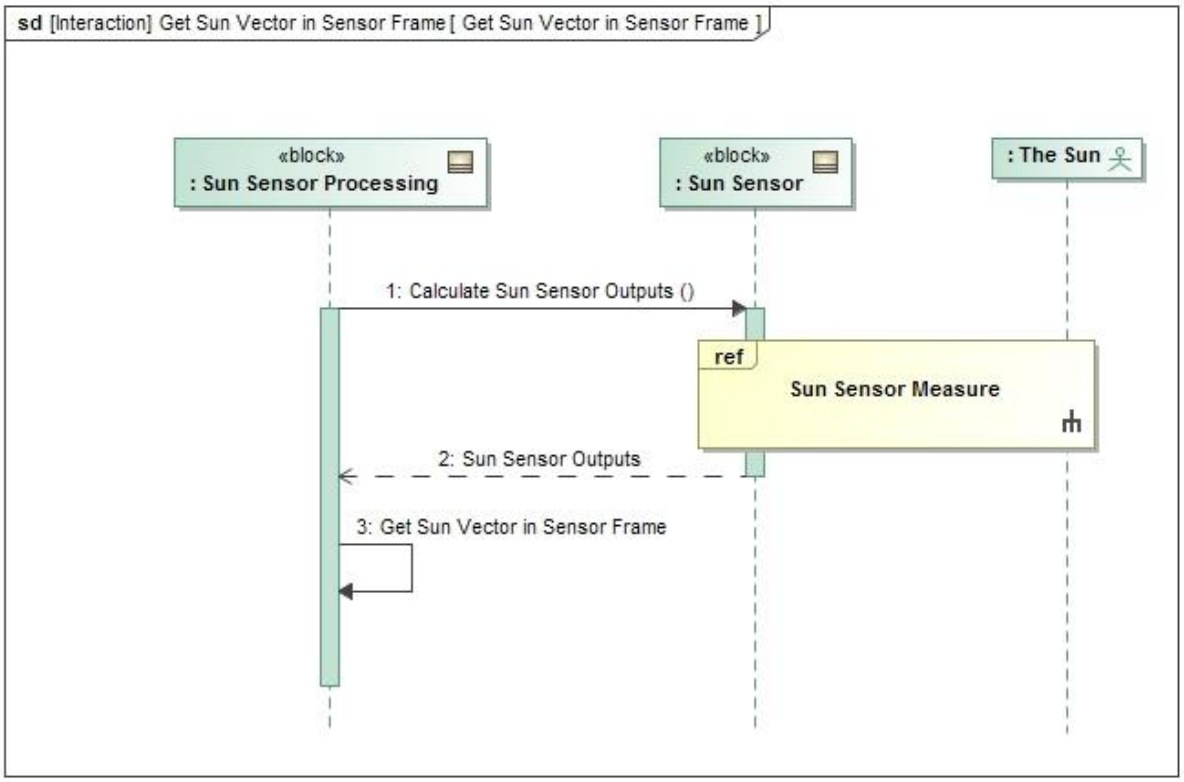

Figure III.3.5.2. The sequence diagram “Get Sun Vector in Sensor Frame”

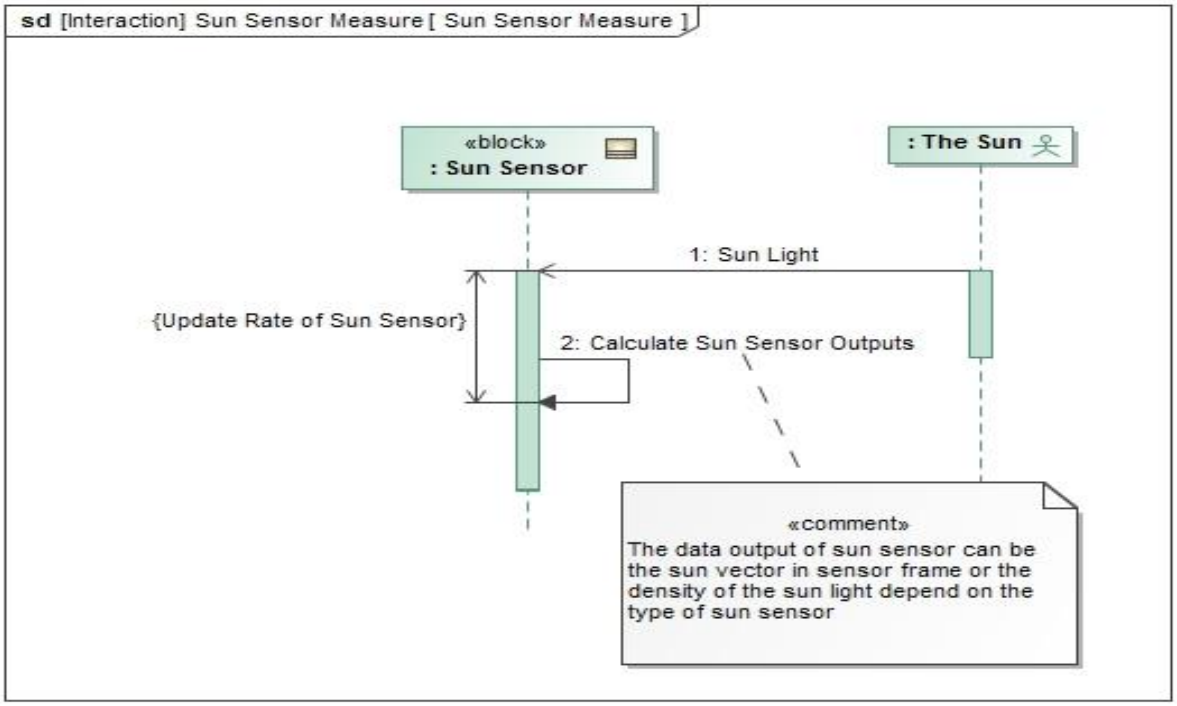

Figure III.3.5.3. The sequence diagram "Sun Sensor Measure"

The activity diagram of “Sun Sensor Processing”

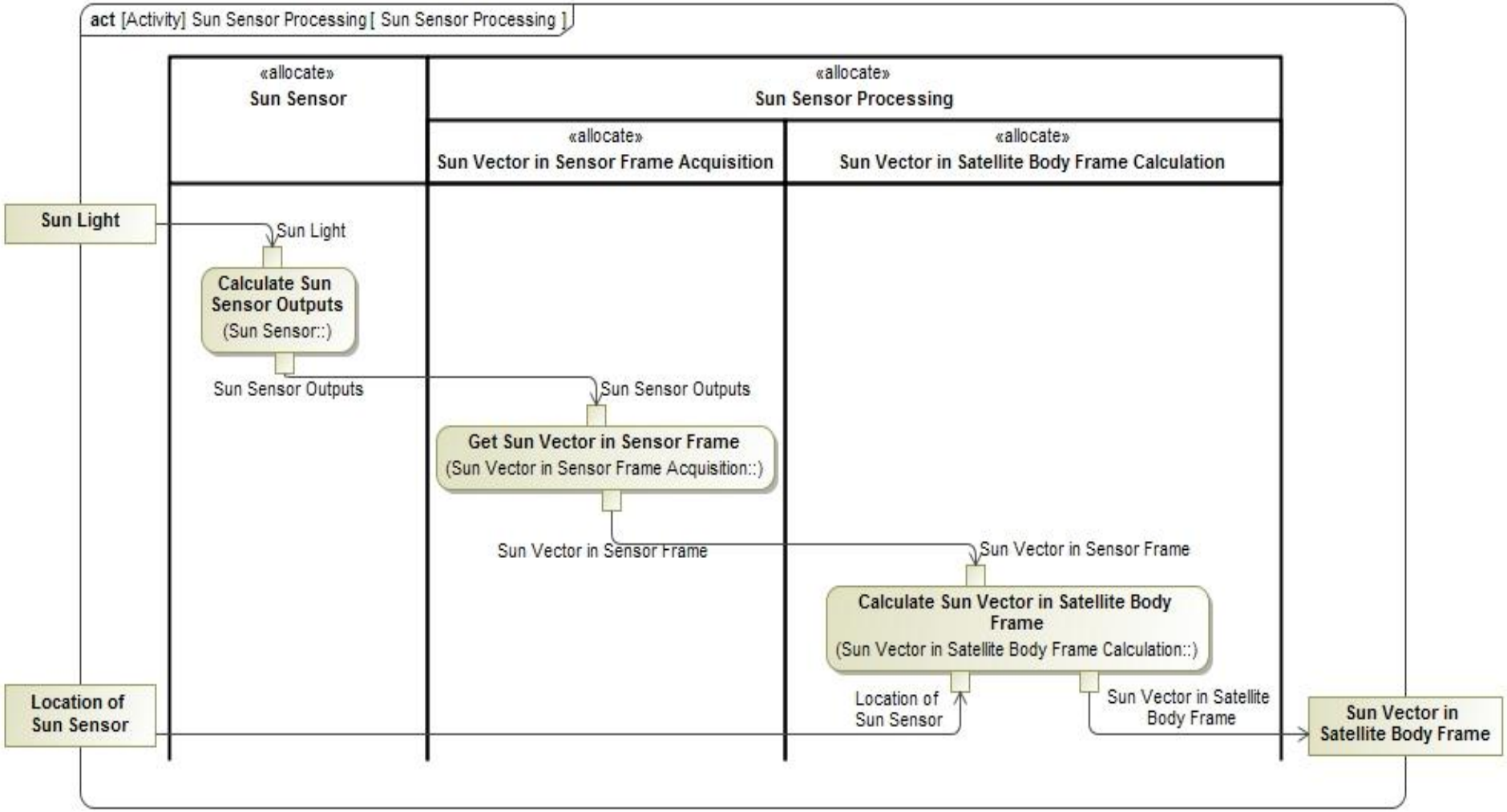

Figure III.3.5.4. The activity diagram of “Sun Sensor Processing”

The internal block diagram of “Sun Sensor Processing”

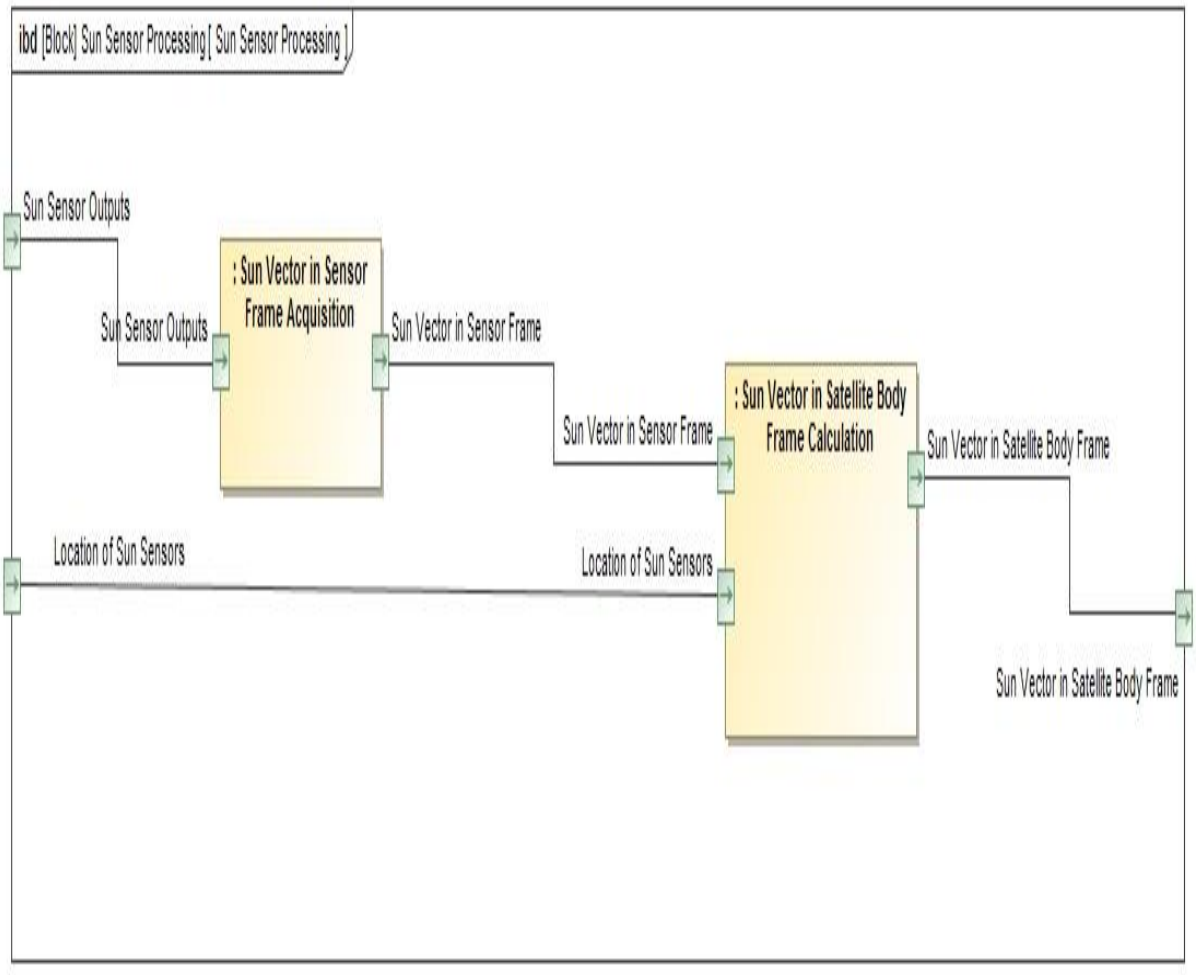

Figure III.3.5.5. The internal block diagram of “Sun Sensor Processing”

III.4. Design of software components of reference vectors estimation module

III.4.1. Reference Magnetic Vector in ECI Estimation

The sequence diagram of "Calculate Magnetic Vector in ECI from Earth Magnetic Model"

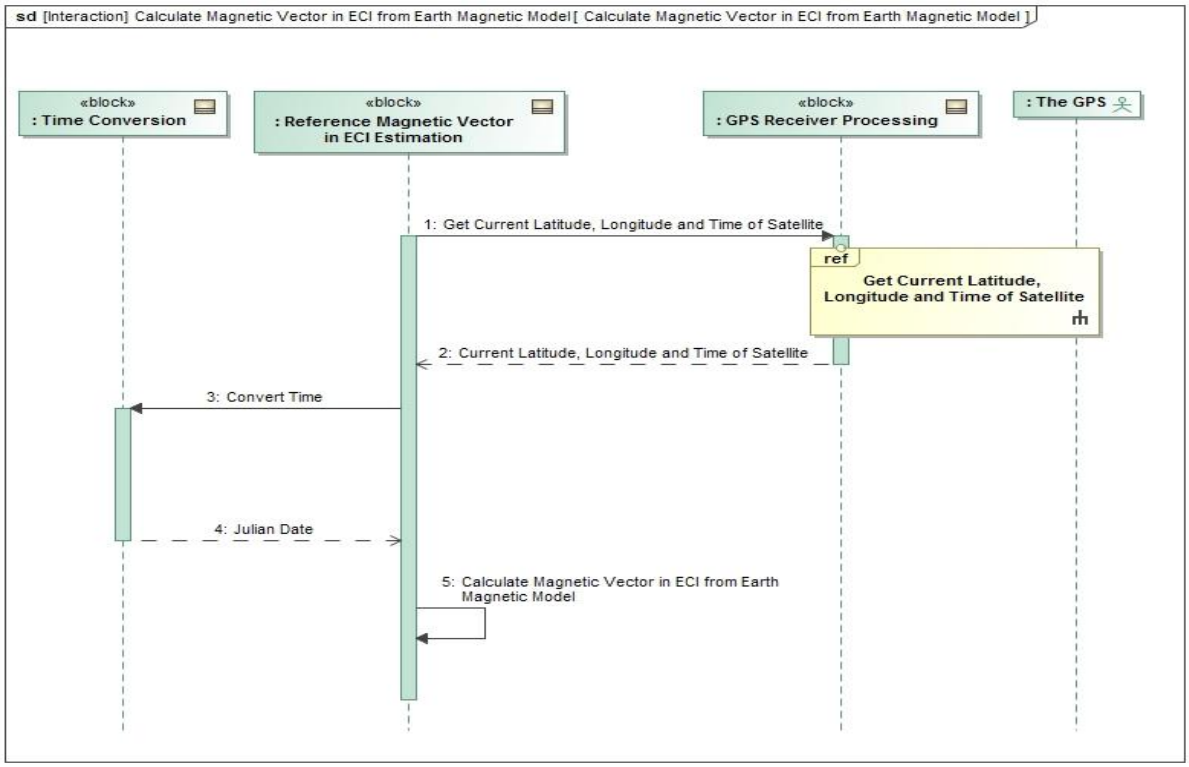

Figure III.4.1.1. The sequence diagram of "Calculate Magnetic Vector in ECI from Earth Magnetic Model"

III.4.2. Reference Sun Vector in ECI Estimation

The sequence diagram of "Calculate Sun Vector in ECI from Sun Model"

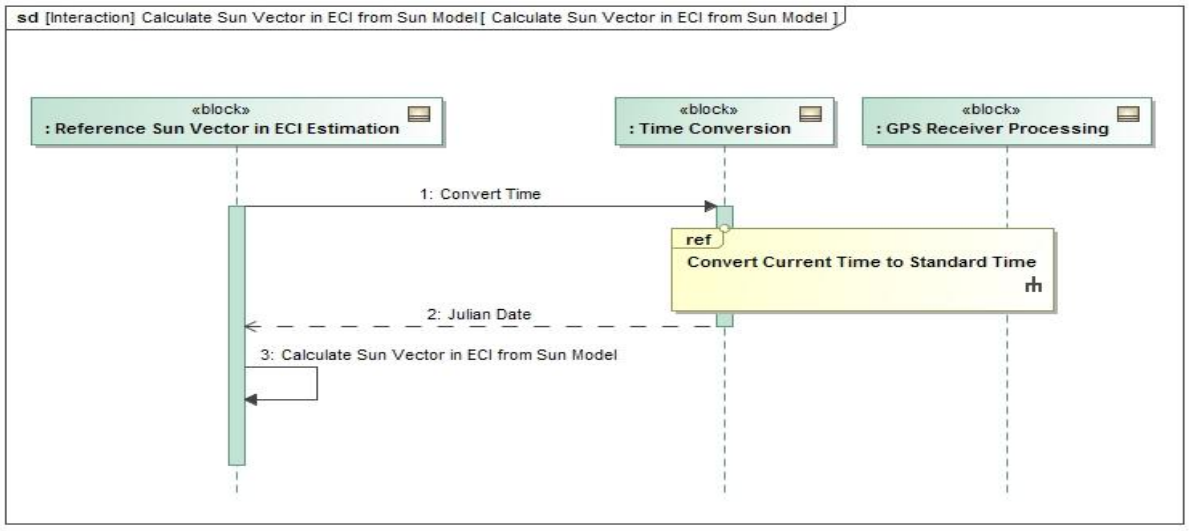

Figure III.4.2.1. The sequence diagram of "Calculate Sun Vector in ECI from Sun Model"

III.4.3. Time Conversion

The sequence diagram of “Convert Current Time to Standard Time”

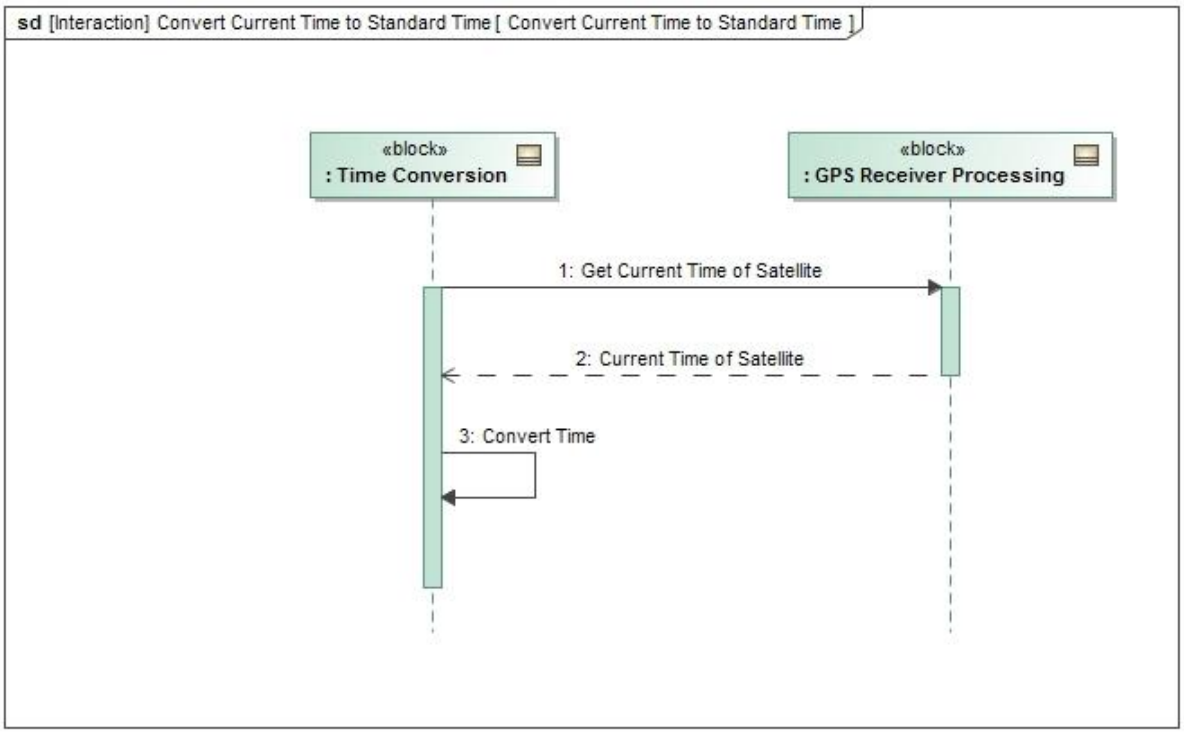

Figure III.4.3.1. The sequence diagram of “Convert Current Time to Standard Time”

III.5.Design of software components of attitude estimation module

The sequence diagram of "Attitude Estimation by TRIAD"

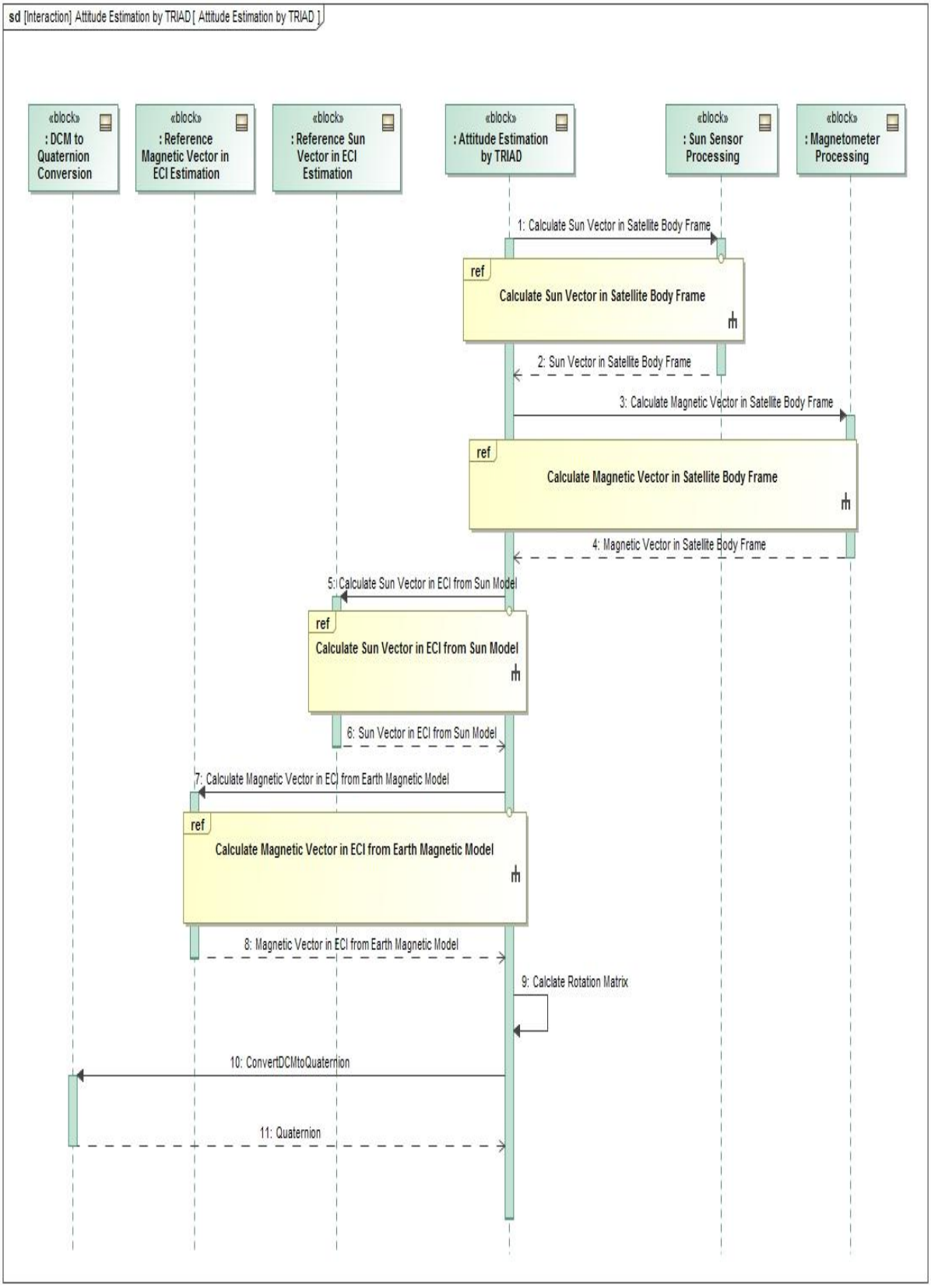

Figure III.5.1. The sequence diagram of "Attitude Estimation by TRIAD"

The sequence diagram of “Kalman Filter by TRIAD and Gyro”

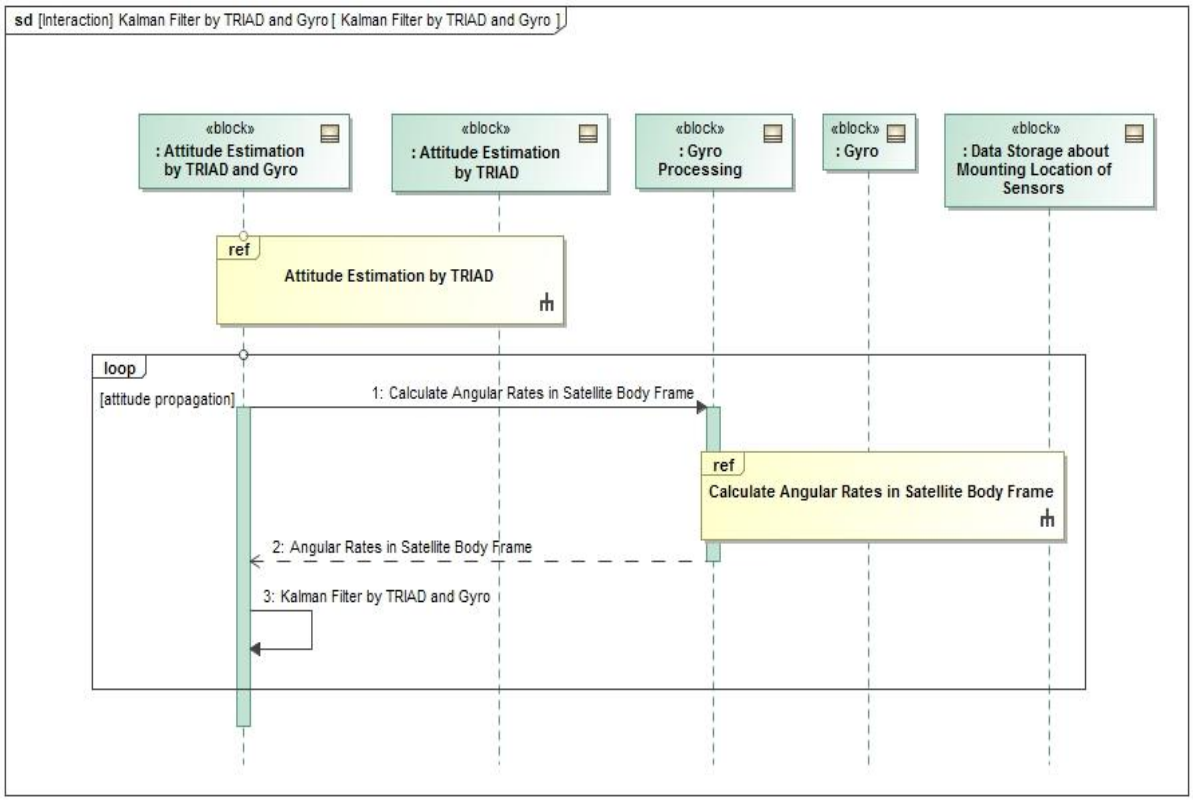

Figure III.5.2. The sequence diagram of “Kalman Filter by TRIAD and Gyro”

The sequence diagram of “Kalman Filter by Star Tracker and Gyro”

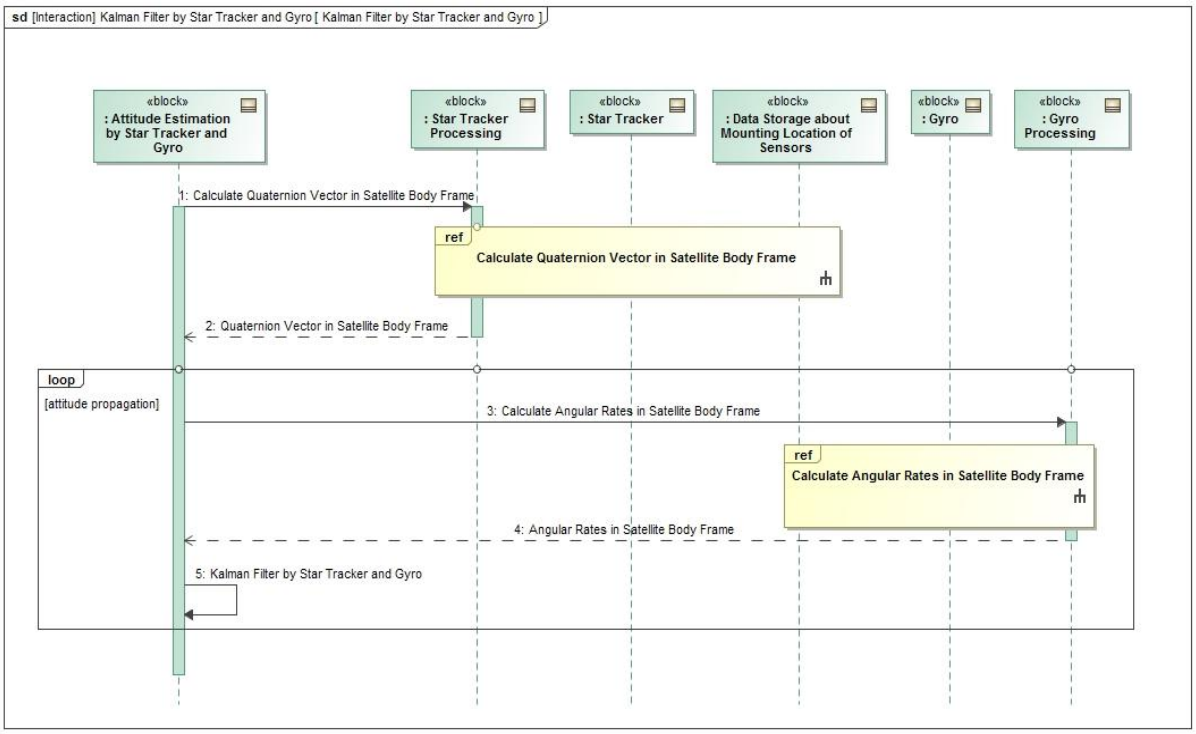

Figure III.5.3. The sequence diagram of “Kalman Filter by Star Tracker and Gyro”

III.6. The modularity and extensibility in design

The modularity is the decomposition of the attitude determination software. The extensibility is the ability to change the sensors and performance of on-board computer of micro/nano satellites projects. For example, there are 6 Sun sensors, 1 Star tracker in Micro Dragon satellite project; however, there is no Star tracker in Nano satellite project, and the performance of onboard computer of Nano satellite is lower. The modularity and extensibility has a close relationship with each other. The modularity is also to support extensibility. There are 3 levels of modularity and extensibility.

At level 1, based on functional analysis, the attitude determination software is decomposed into software modules including sensors processing module, reference vectors estimation module, attitude estimation module and attitude representation conversion module. The sensor processing module is to process the data outputs of sensors. The reference vectors estimation module is to calculate the reference vectors in Earth-Centered Inertial (ECI) frame. The attitude estimation module contains the algorithms for attitude estimation which will be which can be selected based on different accuracy requirements and hardware constraints of each micro/nano satellites projects. The attitude representation conversion module is to convert between the representations of attitude.

Mission dependent parameters such as mounting location of sensors and update rate of sensors are separated in isolated database which can be easily modified when reuse. This decomposition shows the extensibility in design. If the satellite projects are different on the sensor mounting location (on the surface of the satellites) or the updated rates of sensors, only this database need to be modified, the other software modules do not need to be modified.

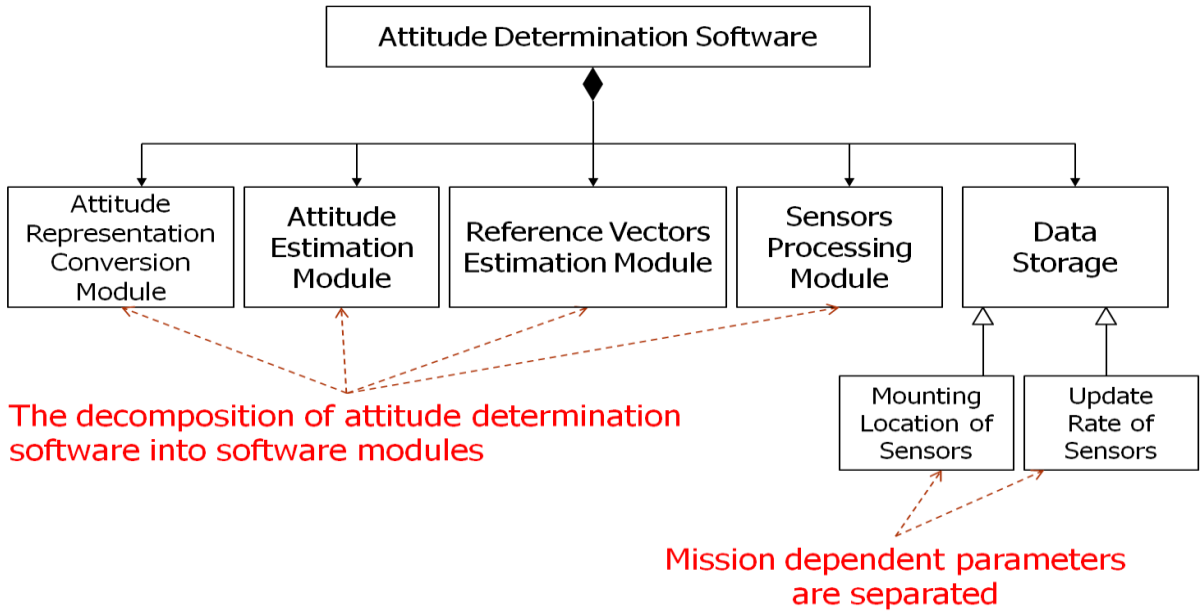

Figure III.6.1. The decomposition of attitude determination software into software modules

At level 2, each software module is decomposed into software components. For example, Figure III.6.2 shows the decomposition of attitude estimation module into software components based on types of sensors. By this decomposition, the attitude estimation components are selected based on sensors when reuse. This decomposition shows the extensibility because the satellites projects are different on used sensors. The nano satellites do not have Star tracker, therefore, without this decomposition, the attitude estimation module cannot be reuse for nano satellites. By this decomposition, the nano satellites can select two software components including attitude estimation by TRIAD and attitude estimation by TRIAD and Gyro when reuse.

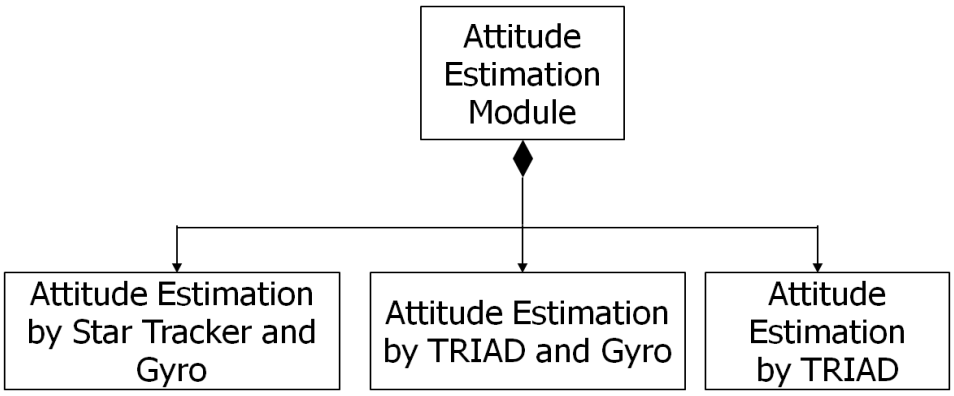

Figure III.6.2. The decomposition of attitude estimation module into software components

Another example is the decomposition of reference vectors estimation module into software components in Figure III.6.3.

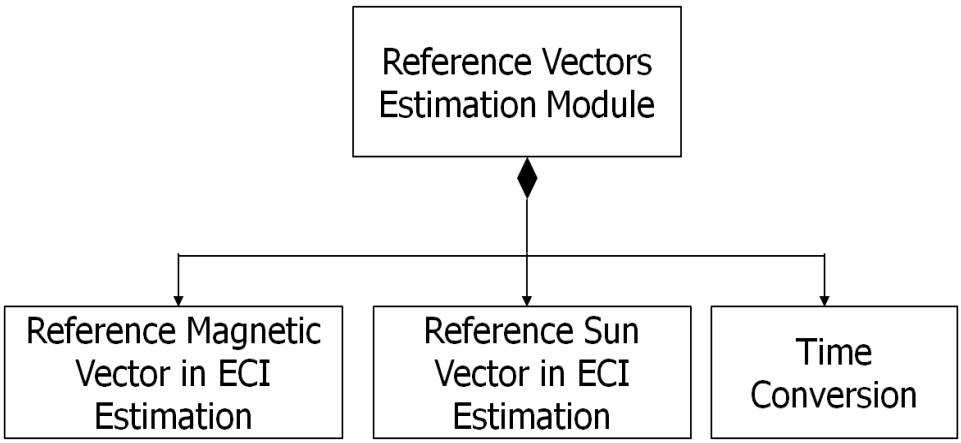

Figure III.6.3. The decomposition of reference vectors estimation module

By this way of decomposition, the reference magnetic vector estimation component, which requires a lot of calculation, can be replaced when reuse based on performance of on-

board computer. For example, with nano satellites, the on-board computers are limited on power of processing, therefore, the simpler version of the reference magnetic vector estimation will be chosen.

Finally, at level 3, each software component is decomposed into basic software components. The basic software components are not decomposed any more in design. Figure III.6.4. is the decomposition of sun sensor processing component into basic software components.

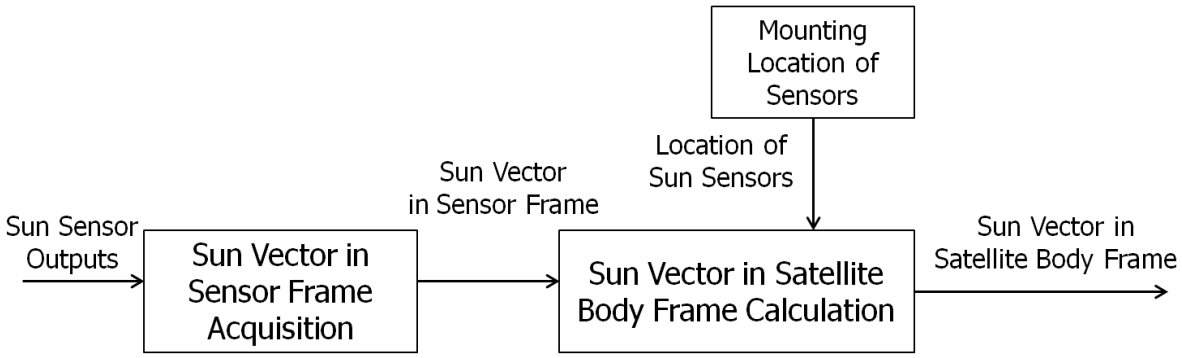

Figure III.6.4. The decomposition of sun sensor processing component

By this way of decomposition, input and output interfaces of basic software components are analyzed when reuse and this shows the extensibility of design. For example, the Sun sensor outputs can be different if the Sun sensor is changed, therefore, the "Sun Vector in Sensor Frame Acquisition" need to be modified when reuse.

IV. Verification and Validation

IV.1.Verification

In this research, to verify the reusability of attitude determination software, the software reuse ratio is defined to measure the percentages of basic software components which can be reused between satellite projects.

Steps to calculate the software reuse ratio for the new satellite project:

1. Calculate number of existing basic software components
2. List all basic software components need to be add for the new satellite (*number of added components*)
3. List all basic software components need to be change for the new satellite (*number of changed components*)
4. Calculate the software reuse ratio by the equation below

$$The\ software\ reuse\ ratio = 1 - \frac{number\ of\ added\ components + number\ of\ changed\ components}{number\ of\ added\ components\ +\ number\ of\ existing\ components}$$

$$= \frac{number\ of\ unchanged\ components}{number\ of\ all\ components}$$

Example 1:

The total number of existing components = 10

The number of new components = 3

The number of change components = 2

The software reuse ratio = 1 - (3+2)/(10+3) = 1 - 5/13 = 8/13

Example 2:

The total number of existing components = 10

The number of new components = 3

The number of change components = 6

The software reuse ratio = 1 - (3+6)/(10+3) = 1-9/13 = 4/13

From the draft estimation, at least 60% of basic software components designed in this research can be reused for the new satellite projects without the need of modification.

In this thesis, there are 16 basic software components have been designed in total. All these basic software components can be used for MicroDragon satellite. There are 10/16 basic software components can be used for Nano Satellite (about 60%).

Table IV.1.1 All designed software components

Sensors Processing Module	
Sun Sensor Processing	
1	Sun Vector in Sensor Frame Acquisition
2	Sun Vector in Satellite Body Frame Calculation
Magnetometer Processing	
3	Magnetic Vector in Sensor Frame Acquisition
4	Magnetic Vector in Satellite Body Frame Calculation
GPS Receiver Processing	
5	Get Current Latitude, Longitude and Time of Satellite
Gyro Processing	
6	Angular Rates in Gyro Frame Acquisition
7	Angular Rates in Satellite Body Frame Calculation
Star Tracker Processing	
8	Quaternion Vector in Sensor Frame Acquisition
9	Quaternion Vector in Satellite Body Frame Calculation
Reference Vectors Estimation Module	
10	Time Conversion
11	Reference Sun Vector in ECI Estimation
12	Reference Magnetic Vector in ECI Estimation
Attitude Estimation Module	
13	Attitude Estimation by TRIAD
14	Attitude Estimation by TRIAD and Gyro
15	Attitude Estimation by Star Tracker and Gyro
Attitude Representation Conversion Module	
16	DCM to Quaternion Conversion

The reusability of the designed attitude determination software is also verified by analyzing SysML diagrams by checking the input and output interfaces of software components. There are only input and output interfaces between software components. Therefore, these software components are reusable when the input and output interfaces are the same.

IV.2. Validation

In this research, the design of reusable attitude determination software needs to validate whether it can save time and cost of satellite development. By analyzing the situation of reuse attitude determination software from MicroDragon project (the current satellite project of Vietnamese students) to next satellite project which is the nano satellite (based on the assumption from Table I.1.2) the goal of saving time and cost of satellite development is proved because the reusable software components will not need time to develop and test again.

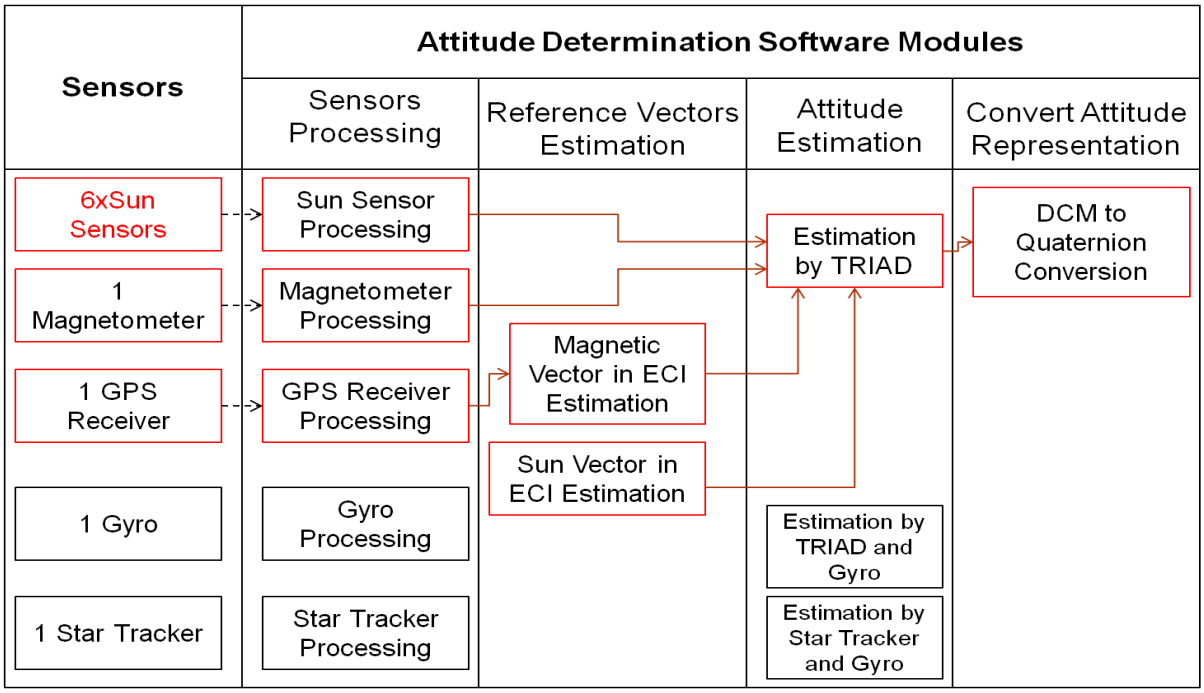

Figure IV.2.1. Attitude determination at Standby mode in Sunshine time for MicroDragon

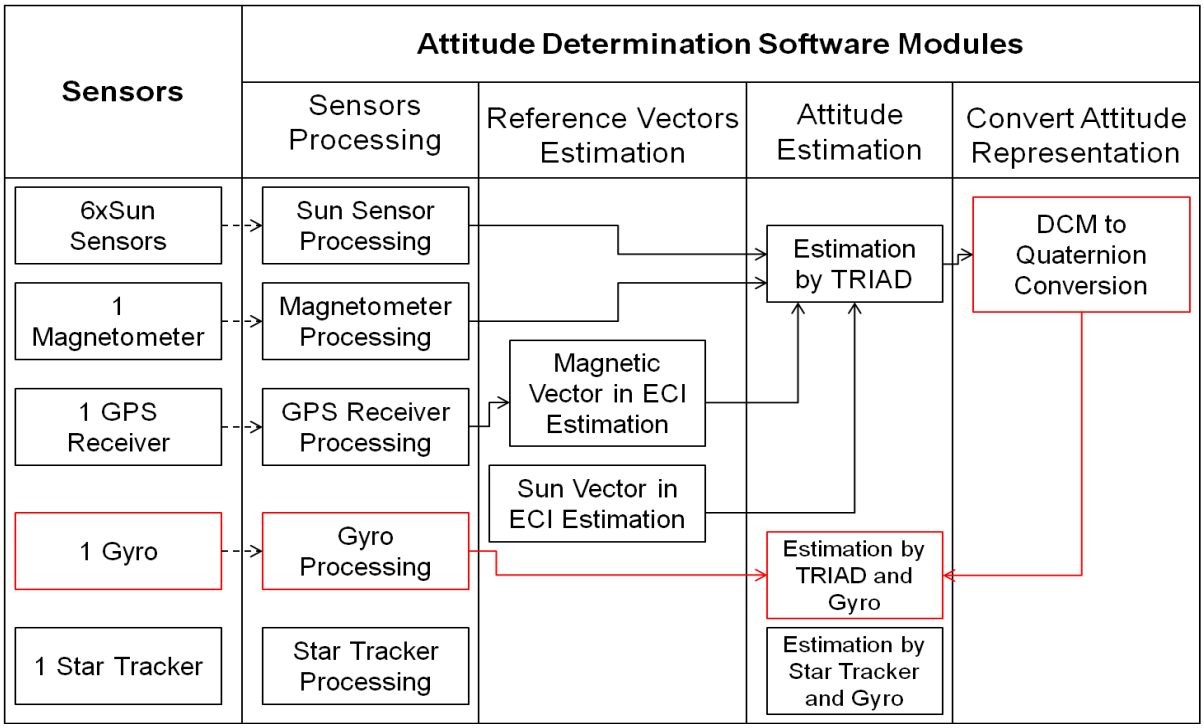

Figure IV.2.2. Attitude determination at Standby mode in Eclipse time for MicroDragon

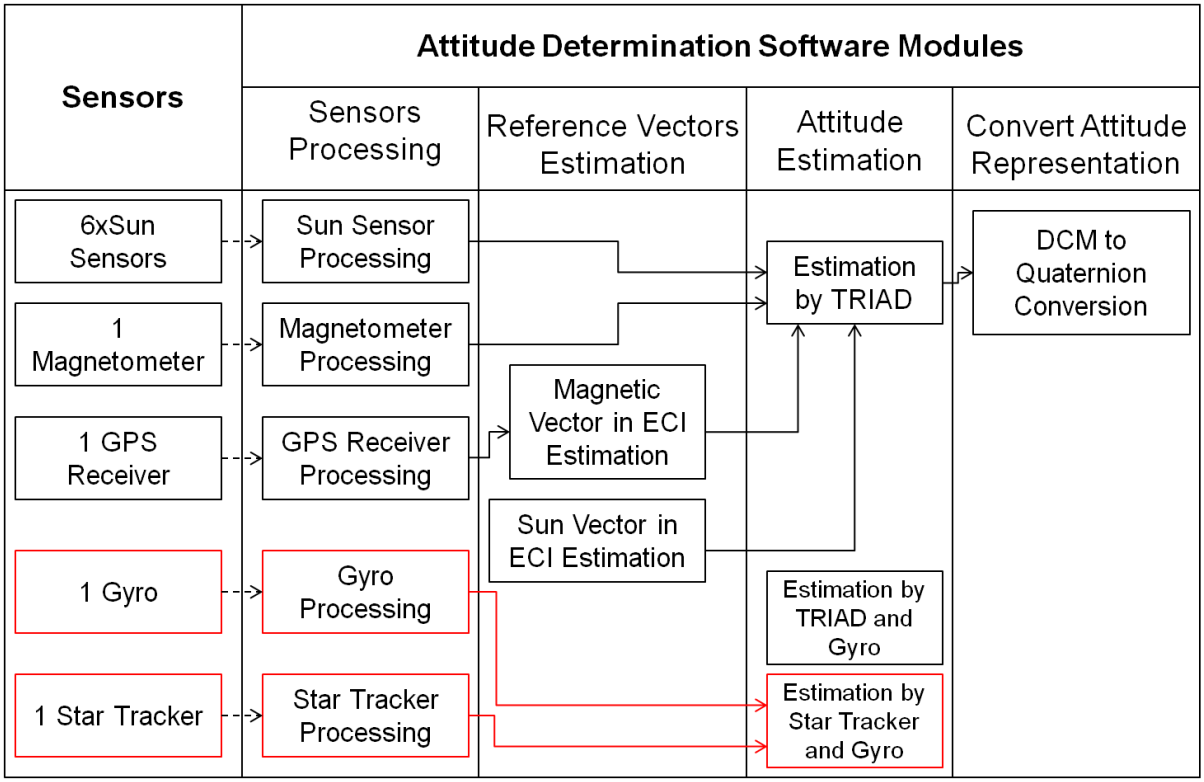

Figure IV.2.3. Attitude determination at Mission mode for MicroDragon

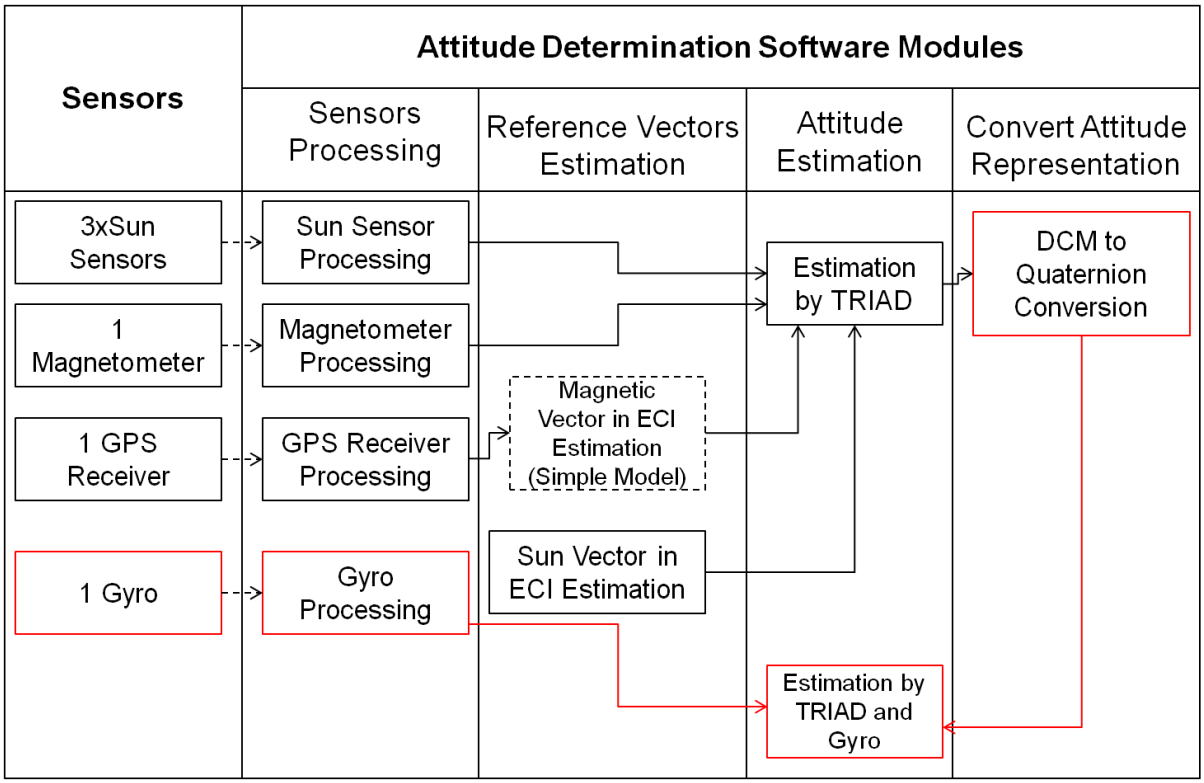

Figure IV.2.4. Attitude determination at Standby mode and Mission mode for Nano Satellite

V. Conclusion

V.1. Summary

There are three key points in this research. Firstly, the reusable attitude determination software is designed by using SysML which has outstanding benefits compare with other methods. Secondly, based on analyzing the difference between micro and nano satellites projects in term of mission and hardware, the viewpoints of modularity and extensibility from NASA RRLs are selected and implemented in my design. Finally, in this research, the designed software is highly modular and extensible.

The results of this research include the design activities and the design of reusable attitude determination software using SysML. In this research, the software modules are designed based on functional analysis and also to support extensibility. The extensibility is the ability to adapt to the change of sensors model and performance of onboard computer. When comparing with levels in NASA RRLs, the modularity achieves level 7, the extensibility achieves level 6.

In this thesis, the libraries of reusable functions in C code for attitude determination including vector, quaternion and matrix calculations; conversion functions of attitude representations and the implementation of TRIAD algorithm are also developed and listed in the Appendix.

By interviewing experts and analyzing the situation of reuse attitude determination software for MicroDragon and Nano satellite project, the designed software can significantly save time and cost of micro/nano satellites development.

V.2. Future works

From the interviews to validate this research, there are mix feelings from the interviewees about the utilizing SysML to design software for satellite's systems. Almost experts agree that design by using SysML has many advantages which are critical for design software reuse of onboard software of satellites. However, there are concerns about the popularity of SysML among the satellite developers. Although, SysML is being increasingly used to model complex systems, the necessary as well as the benefits of applying SysML for designing satellites should be more persuaded to the satellite developers.

Besides, the software tool used to design in this research is Cameo Enterprise Architecture 18.1 which is very powerful tool and its capabilities should be more effectively exploited to design.

References

[1] D. D. Elizabeth Buchen, "2014 Nano / Microsatellite Market Assessment," SpaceWorks Enterprises, Inc. , 2014.

[2] "Small Sat 2014 Conference," [Online]. Available: http://digitalcommons.usu.edu/smallsat/2014/.

[3] E. Buchen, "2015 Small Satellite Market Observations," 2015. [Online]. Available: http://www.spaceworksforecast.com/docs/SpaceWorks_Small_Satellite_Market_Observations_2015.pdf. [Accessed 07 2015].

[4] L. F. L. M. a. M. D. S. E. J., "Kalman Filtering for Spacecraft Attitude Estimation," *Journal of Guidance, Control, and Dynamics,* vol. 5, no. 5, pp. 417-429, 1982.

[5] C. F. L. M. a. Y. C. J. L., "Survey of Nonlinear Attitude Estimation Methods," *AIAA Journal of. Guidance, Control and Dynamics ,* vol. 30, no. 1, pp. 12-28, 2007.

[6] D. F. E. J. J. P. James R. Wertz, Space Mission Engineering: The New SMAD, Microcosm Press, 2011.

[7] S. Nakasuka, "Micro/Nano-satellites On-board Software Framework Design and Its Implementation in Hodoyoshi Satellites," 2013.

[8] W. a. K. K. Frakes, "Software Reuse Research: Status and Future," *IEEE Transactions on Software Engineering,* vol. 31, no. 7, pp. 529-536, 2005.

[9] "Software Reuse FAQ," [Online]. Available: https://earthdata.nasa.gov/esdswg/software-reuse-srwg/software-reuse-faq.

[10] "Tools to Support the Reuse of Software Assets for the NASA Earth Science Decadal Survey Missions," [Online]. Available: http://gsfcir.gsfc.nasa.gov/download/authors/18943/Journal%20Articles_18943.

[11] W. a. C. T. Frakes, "Software Reuse: Metrics and Models," *ACM Computing Surveys,* vol. 28, no. 2, pp. 415-435, 1996.

[12] "Reuse Readiness Levels (RRLs), Version 1.0," [Online]. Available: https://earthdata.nasa.gov/sites/default/files/esdswg/reuse/Resources/rrls/RRLs_v1.0.pdf.

[13] Marshall, J.J.; Downs, R.R., "Reuse Readiness Levels as a Measure of Software Reusability," in *Geoscience and Remote Sensing Symposium, IGARSS, IEEE International Conference*, 2008.

[14] A. M. R. S. Sanford Friedenthal, A practical guide to SysML: the systems modeling language, Second edition ed., Elsevier Inc, 2012.

[15] J. S.Poulin, Measuring Software Reuse: Principles, Practices, and Economic Models, Addison Wesley Longman, Inc., 1997.

[16] "NASA Studying the Reuse of Spacecraft Software," [Online]. Available: http://www.space.com/2304-nasa-studying-reuse-spacecraft-software.html.

List of Figures

List of Tables

Appendices

Appendix1. Listing C source code of reusable functions for attitude determination

//Reusable utilities

//1. Mathematical utilities

1. Vector.h

//This file constains basic functions related to vector calculations

//The default dimention of vector is 3

//Verified 05/09/2015

```
//calc norm of vector
float NormOfVector(float *v)
{
   float norm = sqrt(v[0]*v[0] + v[1]*v[1] + v[2]*v[2]);
   return norm;
}

//Normalize vector
void NormalizeVector(float *v)
{
   float norm = NormOfVector(v);
   if(norm>0){
            v[0] = v[0]/norm;
        v[1] = v[1]/norm;
        v[2] = v[2]/norm;
   }
}

//calc dot product of 2 vectors a and b.
float DotProductOf2Vector(float *a, float *b)
{
        return (a[0]*b[0] + a[1]*b[1] + a[2]*b[2]);
}

//The Cross Product c = a × b of two vectors a and b
```

```
//Reference: https://www.mathsisfun.com/algebra/vectors-cross-product.html
//a × b = |a| |b| sin(a,b)
void VectorCrossProduct(float *a, float *b, float *c)
{
   c[0] = a[1]*b[2] - a[2]*b[1];
   c[1] = a[2]*b[0] - a[0]*b[2];
   c[2] = a[0]*b[1] - a[1]*b[0];
}

//check if 2 vectors are parallel; 1: parallel  0: not parallel
int CheckParallelVector(float *a, float *b)
{
        float *c;
        VectorCrossProduct(a,b,c);
        //if 2 vectors parallel
        if(NormOfVector(c)==0) return 1;
        //if not parallel
        return 0;

}

//calc angular between 2 vectors a and b
//angle output in radian
float AngleOf2Vector(float *a, float *b)
{
        float c;
        c=DotProductOf2Vector(a,b)/(NormOfVector(a)*NormOfVector(b));
        return acos(c);
}
```

2. Quaternion.h

```
//Version 1.2:
//Date 05/08/2015
//These are the basic functions  related to quaternion calculations
//All these functions are reusable
//The users need only call these functions with their parameters
//Reference from:
//http://jp.mathworks.com/help/aeroblks/math-operations.html
//q[0] is scalar
//Normalize quaternion
void NormalizeQuaternion(float *q){
   float norm;  //norm of quaternion
   norm = sqrt(q[0]*q[0] + q[1]*q[1] + q[2]*q[2] + q[3]*q[3]);
   //Normalize quaternion
   if(norm>0)
   {
         q[0] = q[0]/norm;
         q[1] = q[1]/norm;
         q[2] = q[2]/norm;
         q[3] = q[3]/norm;
   }
}
//Properize quaternion
void ProperizeQuatternion(float *q){
         //q0 should alway be positive
   if(q[0]<0.0){          //if q0<0 make q0 be positive
      q[0]*=-1.0;
      q[1]*=-1.0;
      q[2]*=-1.0;
      q[3]*=-1.0;
```

```
    }
}
//Conjugate quaternion
void ConjugateQuaternion(float *q, float *q_conjugated)
{
        q_conjugated[0] = q[0];
        q_conjugated[1] = -q[1];
        q_conjugated[2] = -q[2];
        q_conjugated[3] = -q[3];
}
//Inverse quaternion
void InverseQuaternion(float *q, float *q_inversed){
        //ConjugateQuaternion
        ConjugateQuaternion(q,q_inversed);
        //Normalize quaternion
        NormalizeQuaternion(q_inversed);
}
//Calculate product of two input quaternion
void QuaternionMultiplication(float *q, float *r, float *t)
{
        //output quaternion t = qxr
        t[0] = r[0]*q[0]-r[1]*q[1]-r[2]*q[2]-r[3]*q[3];
        t[1] = r[0]*q[1]+r[1]*q[0]-r[2]*q[3]+r[3]*q[2];
        t[2] = r[0]*q[2]+r[1]*q[3]+r[2]*q[0]-r[3]*q[1];
        t[3] = r[0]*q[3]-r[1]*q[2]+r[2]*q[1]+r[3]*q[0];
}
```

3. Matrix.h

```
#define MAX 3          //number of matrix row, colume
void TransposeMatrix(float A[MAX][MAX], float B[MAX][MAX])
{
                int i,j;
                float C[MAX][MAX];
                for(i=0;i<MAX;i++)
                        for(j=0;j<MAX;j++) C[i][j]=A[j][i];
                for(i=0;i<MAX;i++)
                        for(j=0;j<MAX;j++) B[i][j]=C[i][j];
}
void MultiMatrix(float A[][MAX], float B[][MAX], float C[][MAX])
{
   int i, j, k;
   for(i = 0; i < MAX; i++)
         {
      for(j = 0; j < MAX; j++)
                  {
         C[i][j] = 0.0;
         for(k = 0; k < MAX; k++){
            C[i][j] += A[i][k] * B[k][j];
         }
      }
   }
}
void AddMatrix(float A[MAX][MAX], float B[MAX][MAX], float C[MAX][MAX]){
   int i, j;
   for(i = 0; i < MAX; i++)
            for(j = 0; j < MAX; j++)    C[i][j] = A[i][j] + B[i][j];
}
```

4. TimeConversion.h

// Converse universal time to standard time (Julian date)

//Given year, month, day, hour, minute, second, compute the Julian date, JD

```
float ConverseTime2JulianDate(int year, int month, int day, int hour, int minute, int second)
{
        //reference
        //http://en.wikipedia.org/wiki/Julian_day
        float  a, y, m, jdn, jd;

        a=int((14-month)/12);

        y= year + 4800 - a;
        m= month + 12*a -3;

        //starting from a Gregorian calendar date
        jdn = day + int((153*m+2)/5) + 365*y + int(y/4) - int(y/100) + int(y/400) - 32045;
        jd = jdn + (hour-12)/24 + minute/1440 + second/86400;

        return jd;
}
```

5. SunVectorCalculation.h

```
// Calculate reference sun vector in ECI from sun model when input given time (julian date)
//Compute reference sun vector in ECI
//reference from
http://www.acsu.buffalo.edu/~johnc/space_book/sampex_control_chapt7/solar.m
//Verified by compare with matlab results
void ComputeSunUnitVectorInECIFromJD(float jd, float *sun_i)
{
	// Get Julian Date and Centuries
	float d2r = 3.14159265/180;  //pi=3.14159265   deg to radian
	float jd_cent=(jd-2451545)/36525;

	// Mean Longtitude and Other Paramters
	float lam=280.460+36000.771*jd_cent;

	float m_sun=357.5277233+35999.050*jd_cent; //mean longitude of Sun

	float lam_ecl=lam+1.914666471*sin(m_sun*d2r)+0.019994643*sin(2*m_sun*d2r);

	float eps=23.439291-0.0130042*jd_cent;  //ecliptic longitude of Sun

	//reference sun vector in eci  s_eci
	sun_i[0]=cos(lam_ecl*d2r);    //cos lamda ecliptic
	sun_i[1]=cos(eps*d2r)*sin(lam_ecl*d2r);
	sun_i[2]=sin(eps*d2r)*sin(lam_ecl*d2r);
}
```

6. File Triad.h

```
// The Triad algorithm computes DCMbi from s_eci,s_b,m_eci,m_b
//Verified 05/10/2015
//Reference from: http://www.dept.aoe.vt.edu/~cdhall/courses/aoe4140/
void TRIAD(float *v1i, float *v2i, float *v1b, float *v2b, float DCMbi[3][3])
{
        ////v1i, v2i: Sun vector and magnetic field vector in ECI
        //v1b, v2b: Sun vector and magnetic field vector in Body
        float t2i[3], t3i[3], t2b[3], t3b[3];
        float Rbt[3][3], Rti[3][3];

        VectorCrossProduct(v1i,v2i,t2i);
        NormalizeVector(t2i);

        VectorCrossProduct(v1i,t2i,t3i);
        NormalizeVector(t3i);

        VectorCrossProduct(v1b,v2b,t2b);
        NormalizeVector(t2b);

        VectorCrossProduct(v1b,t2b,t3b);
        NormalizeVector(t3b);

        //Construct Rbt
        Rbt[0][0]=v1b[0];      Rbt[0][1]=t2b[0];      Rbt[0][2]=t3b[0];
        Rbt[1][0]=v1b[1];      Rbt[1][1]=t2b[1];      Rbt[1][2]=t3b[1];
        Rbt[2][0]=v1b[2];      Rbt[2][1]=t2b[2];      Rbt[2][2]=t3b[2];

        //Construct Rti
        Rti[0][0]=v1i[0];      Rti[0][1]=t2i[0];      Rti[0][2]=t3i[0];
        Rti[1][0]=v1i[1];      Rti[1][1]=t2i[1];      Rti[1][2]=t3i[1];
        Rti[2][0]=v1i[2];      Rti[2][1]=t2i[2];      Rti[2][2]=t3i[2];

        TransposeMatrix(Rti,Rti);

        MultiMatrix(Rbt,Rti,DCMbi);
}
```

7. ConverseAttitudeRepresentation.h

```
//These are functions which convert between attitude representation
//All these functions are reusable
//Reference from:
//http://jp.mathworks.com/help/aeroblks/axes-transformations.html
//Convert Quaternion to DCM (Direction Cosine Matrix)
//Reference from Prof. Nakasuka Lecturers at Keio University 2013
//DCM=Cb2i  transformation from body to inertia
void ConvertQuaternion2DCMbi(float *q, float DCM[][3]){

        //calculate elements of DCM matrix
   DCM[0][0] = q[0]*q[0] + q[1]*q[1] - q[2]*q[2] - q[3]*q[3];
        DCM[0][1] = 2.0*(q[1]*q[2] - q[0]*q[3]);
        DCM[0][2] = 2.0*(q[1]*q[3] + q[0]*q[2]);

   DCM[1][0] = 2.0*(q[1]*q[2] + q[0]*q[3]);
   DCM[1][1] = q[0]*q[0] - q[1]*q[1] + q[2]*q[2] - q[3]*q[3];
   DCM[1][2] = 2.0*(q[2]*q[3] - q[0]*q[1]);

   DCM[2][0] = 2.0*(q[1]*q[3] - q[0]*q[2]);
   DCM[2][1] = 2.0*(q[2]*q[3]+q[0]*q[1]);
   DCM[2][2] = q[0]*q[0] - q[1]*q[1] - q[2]*q[2] + q[3]*q[3];
}
```

8. File TestADS.cpp

```
//Verification the correctness of functions
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>     /* srand, rand */
#include <time.h>       /* time */
#include <math.h>
//Reusable utilities
//1. Mathematical utilities
#include "Vector.h"
#include "Quaternion.h"
#include "Matrix.h"
//2. Converse universal time to standard time (Julian date)
#include "TimeConversion.h"
//3. Calculate reference sun vector in ECI from sun model when input given time (julian date)
#include "SunVectorCalculation.h"
//4. The Triad algorithm computes DCMbi from s_eci,s_b,m_eci,m_b
#include "Triad.h"
//5. Converse attitude representation
#include "ConverseAttitudeRepresentation.h"

//#define NMAX 4 //for test quaternion
void TestQuaternion()
{
        float q1[4];
        float q2[4];
        float q3[4];

         q1[0] =0.9962;
         q1[1] =  0;
         q1[2] =  0;
         q1[3] = 0.0872;

         q2[0] =0.9962;
         q2[1] =  0;
         q2[2] =  0;
         q2[3] = 0.0872;

        //Test Multiplication
        QuaternionMultiplication(q1,q2,q3);
        //Show results, OK
        printf("%5.3f %5.3f %5.3f %5.3f", q3[0], q3[1], q3[2], q3[3]);
         //0.9848        0        0    0.1736
}
```

```
void TestTimeConversion()
{
	float  jd;
	float sun_eci[3];

	//Test case to check the time conversion function
	int year=2015;
	int month=5;
	int day=8;
	int hour=13;
	int minute=24;
	int second=20;

	jd=ConverseTime2JulianDate(year, month, day, hour, minute, second);

	//Result from online conversion
	//http://aa.usno.navy.mil/data/docs/JulianDate.php
	//The Julian date for CE  2015 May  8 13:24:20.0 UT is
	//JD 2457151.058565

	ComputeSunUnitVectorInECIFromJD(jd, sun_eci);
	printf("%f %f %f", sun_eci[0], sun_eci[1], sun_eci[2]);
}

void TestVector()
{
	float  norm, a[3], b[3], v[3];
	a[0]=4; a[1]=0; a[2]=7;
	b[0]=-2; b[1]=1; b[2]=3;

	//VectorCrossProduct(a,b,v);
	//NormalizeVector(v);

	//int i=CheckParallelVector(a,b);
	//printf("%d",i);

	float c;
	c=AngleOf2Vector(a,b);

	printf("%.2f",c);
	//printf("%.2f %.2f %.2f",v[0], v[1], v[2]);
}
```

```
//Ref: http://www.dept.aoe.vt.edu/~cdhall/courses/aoe4140/
void TestTriad()
{
	////v1i, v2i: Sun vector and Magnetic field vector in ECI
	//v1b, v2b: Sun vector and Magnetic field vector in Body

	float v1i[3], v2i[3], v1b[3], v2b[3];
	float DCMbi[3][3]; //determined attitude

	//test case from http://www.dept.aoe.vt.edu/~cdhall/courses/aoe4140/
	v1i[0]=-0.1517; v1i[1]=-0.9669; v1i[2]=0.2050;
	v2i[0]=-0.8393; v2i[1]= 0.4494; v2i[2]=-0.3044;

	v1b[0]=0.8273; v1b[1]=0.5541; v1b[2]=-0.0920;
	v2b[0]=-0.8285; v2b[1]=0.5522; v2b[2]=-0.0955;

	TRIAD(v1i,v2i,v1b,v2b,DCMbi);

	printf("Test Triad Algorithm:\n");
	int i,j;

	for(i=0;i<3;i++)
		{
			for(j=0;j<3;j++) printf("%.4f ",DCMbi[i][j]);
			printf("\n");
		}
	//result is OK

}
int main(){

	TestQuaternion();
	TestTimeConversion();
	TestVector();

	TestTriad();

	return 0;
```

Appendix 2. Transformation from Sensor Frame to Satellite Body Frame

Define $DCM_{SensorFrame}^{Body}$ is the Direction Cosine Matrix which transform the Sensor Frame to the Satellite Body Frame.

$DCM_{SensorFrame}^{Body}$ is calculated from 3 Euler angles Ψ, θ, Φ and using Z-Y-X rotation sequence from Sensor Frame to the Satellite Body Frame.

Ψ is the rotation angle about the Z-axis of Sensor Frame.

θ is the rotation angle about the Y-axis of Sensor Frame.

Φ is the rotation angle about the X-axis of Sensor Frame.

3 Euler angles Ψ, θ, Φ is measured when the sensor is integrated to the satellite.

For any mounting location of sensors including Sun sensor, Gyro, Magnetometer, the transformation from Sensor Frame to Satellite Body Frame is calculated from the equation (2.1).

$$DCM_{SensorFrame}^{Body} = \begin{bmatrix} \cos\theta.\cos\psi & -\cos\phi.\sin\psi + \sin\phi.\sin\theta.\cos\psi & \sin\phi.\sin\psi + \cos\phi.\sin\theta.\cos\psi \\ \cos\theta.\sin\psi & \cos\phi.\cos\psi + \sin\phi.\sin\theta.\sin\psi & -\sin\phi.\cos\psi + \cos\phi.\sin\theta.\sin\psi \\ -\sin\theta & \sin\phi.\cos\theta & \cos\phi.\cos\theta \end{bmatrix} \quad (2.1)$$

However, for Star tracker, because the output of Star tracker is the quaternion, therefore, to convert quaternion output in Sensor Frame to the quaternion in Satellite Body Frame, the $DCM_{SensorFrame}^{Body}$ should be transformed to the quaternion $q_{SensorFrame}^{Body}$.

Define $DCM_{SensorFrame}^{Body} = \begin{bmatrix} DCM_{11} & DCM_{12} & DCM_{13} \\ DCM_{21} & DCM_{22} & DCM_{23} \\ DCM_{31} & DCM_{32} & DCM_{33} \end{bmatrix}$

The conversion from $DCM_{SensorFrame}^{Body}$ to the $q_{SensorFrame}^{Body}$ is as below.

Define $q_{SensorFrame}^{Body} = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 \end{bmatrix}^T$, q_4 is the scalar.

Firstly, q_1, q_2, q_3 and q_4 are calculated:

$$q_1 = \sqrt{\frac{1}{4}(1 + DCM_{11} - DCM_{22} - DCM_{33})}$$

$$q_2 = \sqrt{\frac{1}{4}(1 - DCM_{11} + DCM_{22} - DCM_{33})}$$

$$q_3 = \sqrt{\frac{1}{4}(1 - DCM_{11} - DCM_{22} + DCM_{33})}$$

$$q_4 = \sqrt{\frac{1}{4}(1 + DCM_{11} + DCM_{22} + DCM_{33})}$$

Secondly, q_1, q_2, q_3 and q_4 are re-calculated by the Table 2.1. For example, if the q_4 is the maximum compare to other elements, the q_1, q_2 and q_3 are re-calculated as the first row of the Table 2.1.

Table 2.1. Calculating $q_{SensorFrame}^{Body}$ elements

Maximum	q_4	q_1	q_2	q_3
q_4	q_4	$\frac{DCM_{32}-DCM_{23}}{4q_4}$	$\frac{DCM_{13}-DCM_{31}}{4q_4}$	$\frac{DCM_{21}-DCM_{12}}{4q_4}$
q_1	$\frac{DCM_{32}-DCM_{23}}{4q_1}$	q_1	$\frac{DCM_{21}+DCM_{12}}{4q_1}$	$\frac{DCM_{13}+DCM_{31}}{4q_1}$
q_2	$\frac{DCM_{13}-DCM_{31}}{4q_2}$	$\frac{DCM_{21}+DCM_{12}}{4q_2}$	q_2	$\frac{DCM_{32}+DCM_{23}}{4q_2}$
q_3	$\frac{DCM_{21}-DCM_{12}}{4q_3}$	$\frac{DCM_{13}+DCM_{31}}{4q_3}$	$\frac{DCM_{32}+DCM_{23}}{4q_3}$	q_3

In brief, the diagram of transformation function from Sensor Frame to Satellite Body Frame is showed in Figure 2.1.

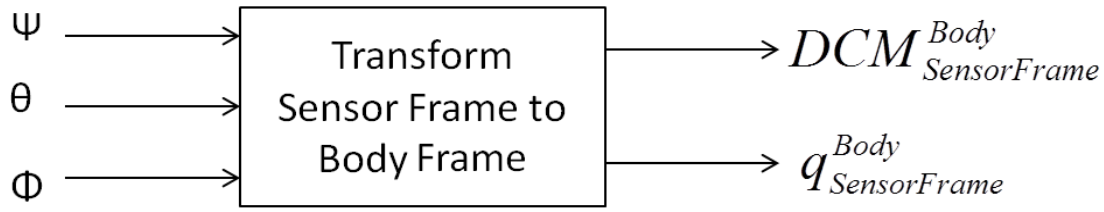

Figure 2.1. Diagram of transform function from Sensor Frame to Satellite Body Frame

For Star tracker, the calculating of the quaternion in the Satellite Body Frame is as below

$$q_{Body} = q_{SensorFrame}^{Body} q_{SensorFrame}$$

q_{Body} is the quaternion in the Satellite Body Frame.

$q_{SensorFrame}$ is the quaternion output in the Star tracker Frame.

For Sun sensor, the calculating of the sun vector in the Satellite Body Frame is below

$$S_{Body} = DCM_{SensorFrame}^{Body} S_{SensorFrame}$$

S_{Body} is the sun vector in the Satellite Body Frame

$S_{SensorFrame}$ is the sun vector measured in the Sun sensor Frame

For Gyro, the Gyro Frame should be aligned with the Satellite Body Frame when integrating, the angular rate vector in Satellite Body Frame is calculated as below

$$\begin{bmatrix} w_x^{Body} \\ w_y^{Body} \\ w_z^{Body} \end{bmatrix} = \begin{bmatrix} w_x^{SensorFrame} \\ w_y^{SensorFrame} \\ w_z^{SensorFrame} \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \\ d_z \end{bmatrix}$$

$w^{Body} = \begin{bmatrix} w_x^{Body} & w_y^{Body} & w_x^{Body} \end{bmatrix}^T$ is the angular rate vector in Satellite Body Frame.

$w^{SensorFrame} = \begin{bmatrix} w_x^{SensorFrame} & w_y^{SensorFrame} & w_z^{SensorFrame} \end{bmatrix}^T$ is the angular rate vector measured in Gyro Frame.

The offset vector $d = [d_x \quad d_y \quad d_z]^T$ is measured during the integration of Gyro to the Satellite.

For GPS Receiver processing, because the GPS Receiver outputs are the location of the sensor, therefore, there is no need to transform the outputs of GPS Receiver to the Satellite Body Frame. The location of Satellite can be considered as the location of the GPS Receiver.

Appendix 3. Simulation of using Kalman Filter

1. Introduction about Kalman Filter

The original Kalman Filter is only applicable for linear systems. However, it is extended to deal with nonlinear systems by Extended Kalman Filter (EKF). The EKF is not always optimal and can diverge if initial errors are too large or if the system model is inaccurate.

Defining the discrete nonlinear system as:

$$x_k = f(x_{k-1}, u_{k-1}) + w_{k-1} \qquad (3.1.1)$$

$$z_k = h(x_k) + v_k \qquad (3.1.2)$$

where:

x is state vector

$f(.)$ describes the system dynamics

u is control input

w is process noise

h is the measurement model

v is the measurement noise

the subscript k denotes discrete time.

Both measurement and process noise are assumed to be zero mean Gaussian. Using the system described above, the equations for EKF:

At Predict phase:

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1})$$

$$P_k^- = F_k P_k F_k^T + Q$$

At Update phase:

$$K_k = P_k^- H_k^T (H_k P_k^- H_k^T + R)^{-1}$$

$$\hat{x}_k = \hat{x}_k^- + K_k (z_k - H_k \hat{x}_k^-)$$

$$P_k = (I - K_k H_k) \mathrm{P}_k^-$$

where:

$\hat{x}$ denotes estimated state vector

P is covariance matrix

K is calculated Kalman gain

$$F_k = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}_{k-1},u}$$

is the derivate of the nonlinear system with respect to the states

$$H_k = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_{k-1}}$$

is the derivate of the measurement equations with respect to the states.

R is the measurement covariance matrix.

Q is the process covariance matrix.

Due to the complexity of the EKF, it is limited when implementation on a on-board computer of nano satellites.

2. Simulation of using Kalman Filter for satellite with one axis

Physical model:

The system model of a satellite with one axis is described in Figure 3.1. The sensors consist of a gyro and a star tracker.

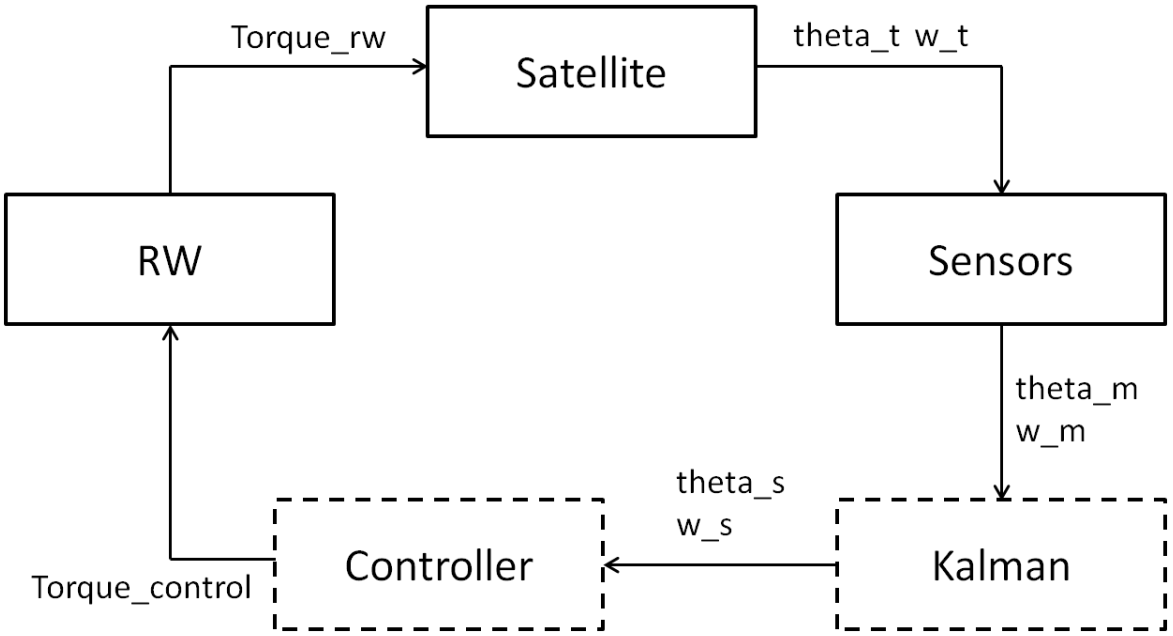

Figure 3.1 System model of satellite with one axis

One axis rotational angle Θ of satellite is measured.

The true value of Θ is: Θ_t

$$\ddot{\Theta}_t = w \qquad (3.2.1)$$

w is white noise with $1\sigma = 0.01$ rad/sec^2

The true value of angular velocity of satellite:

$$w_t = \dot{\Theta}_t \qquad (3.2.2)$$

This angular velocity is measured by gyro at every 10 msec:

$w_m = w_t + w_1 + r$ (3.2.3)

w_1 is white noise with $1\sigma = 0.1$ rad/sec

r is random walk with:

$\dot{r} = w_2$ (3.2.4)

w_2 is white noise with $1\sigma = 0.1$ rad/sec^2

At every 1s, the star sensor (star tracker) is used to measure Θ:

$\Theta_m = \Theta_t + v$ (3.2.5)

with v is white noise $1\sigma = 0.01$ rad

Derive the system dynamic equation and measurement equation

To estimate Θ and r, the estimated of these values are included in state vector **x**:

Define the notation:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad \dot{\mathbf{x}} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix}$$

$x_1 = \Theta_s$

$x_2 = \dot{\Theta}_s$

$x_3 = r_s$

The system estimation model:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} = \begin{bmatrix} 0 & 1 & -1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ w \\ w_2 \end{bmatrix}$$

Define the notation:

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & -1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \ \mathbf{w} = \begin{bmatrix} 0 \\ w \\ w_2 \end{bmatrix}$$

the system dynamic equation:

$$\frac{\mathbf{dx}}{\mathbf{dt}} = \dot{\mathbf{x}} = \mathbf{A}*\mathbf{x} + \mathbf{B}*\mathbf{w} \qquad (3.2.6)$$

Figure 3.2 shows the state vector including theta Θ and angular velocity w without using Kalman Filter. The state of system is not converged.

Figure 3.3 shows the states including theta Θ, angular velocity w and covariance matrix P when using Kalman Filter. The state of system is converged after 6s.

The C source code of the simulation is showed as below.

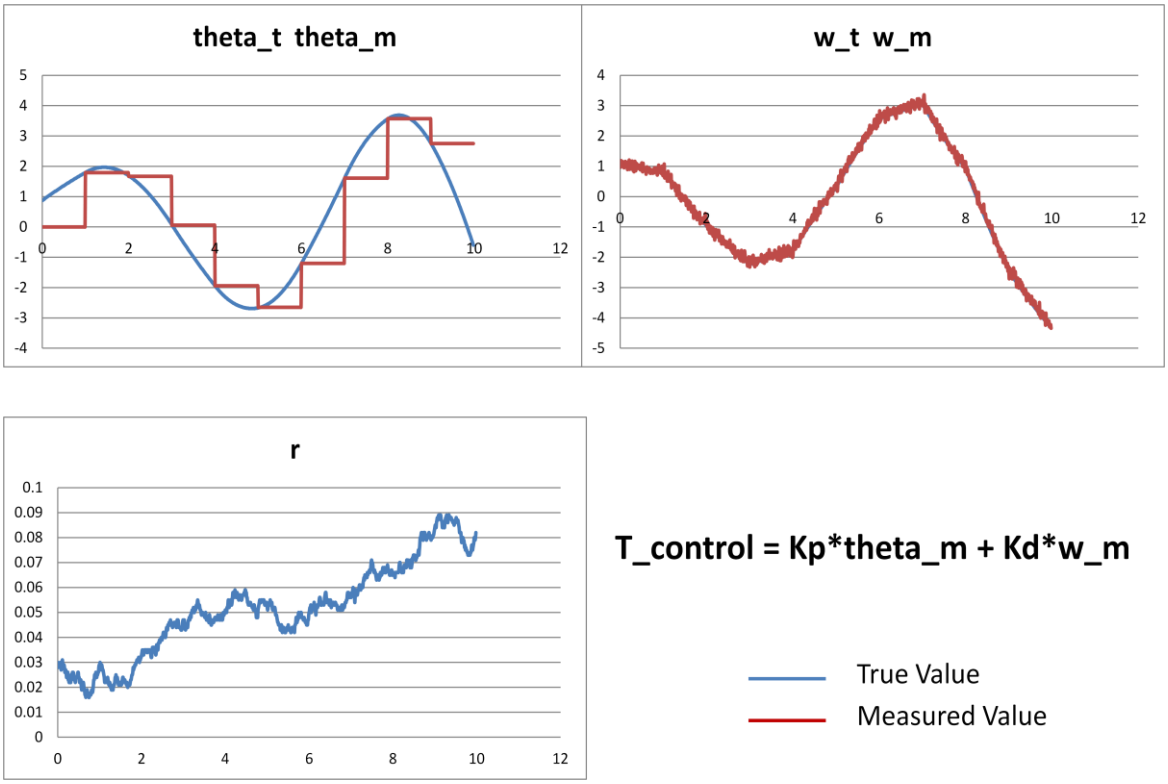

Figure 3.2 States without using Kalman Filter

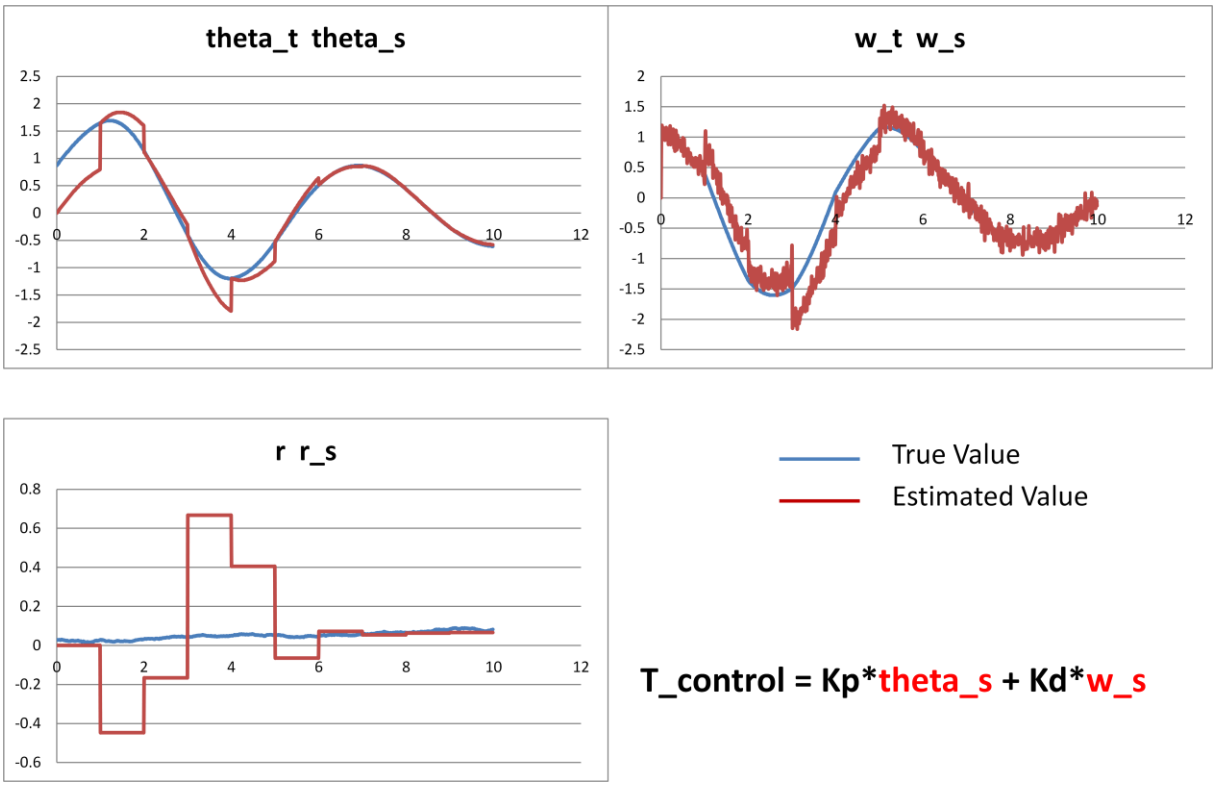

Figure 3.2 States using Kalman Filter

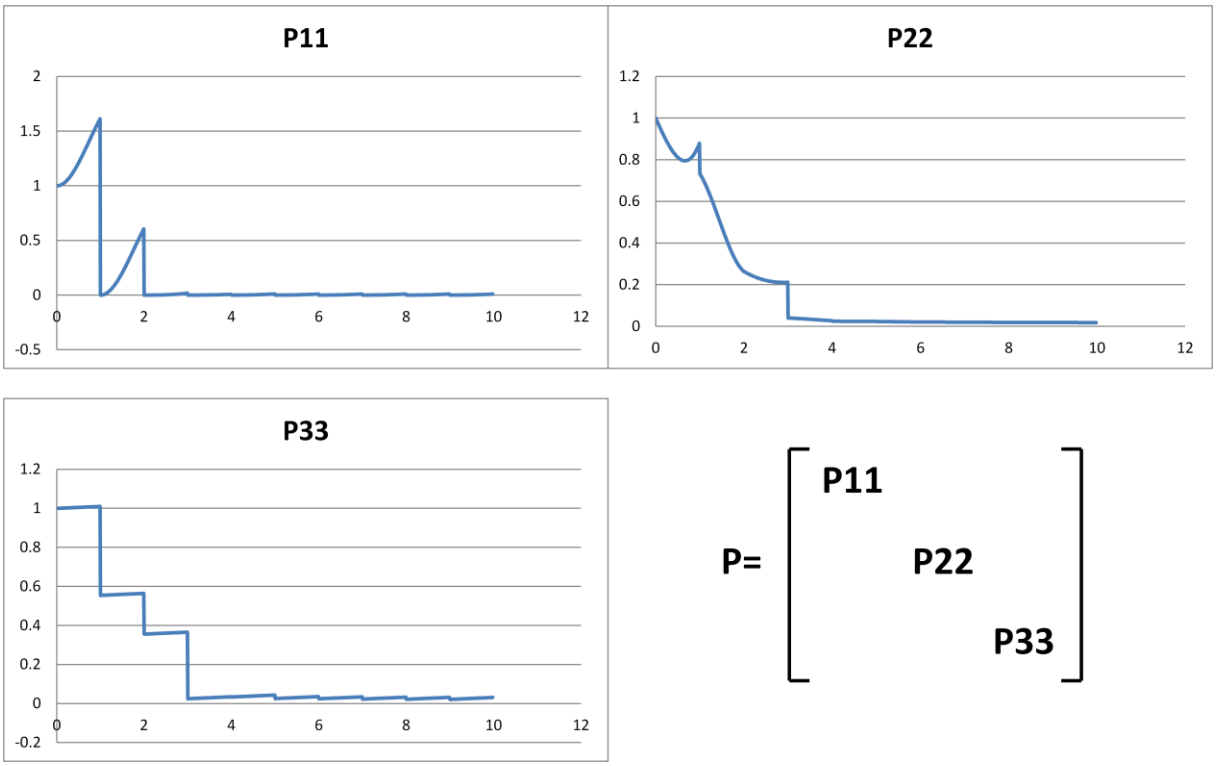

Figure 3.3 State of covariance matrix using Kalman Filter

C source code of simulation Kalman Filter for Gyro and Star tracker

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <stdlib.h>     /* srand, rand */
#include <time.h>       /* time */
#include "Matrix.h"
#include "Noise.h"

#define PI 3.14159265359    //PI
#define NMAX 1000  // number of iterations 10000
#define MAX 3    //number of matrix row, colume

//physical simulation
double dt=0.01;  //dt=10 msec

//----1. Satellite Axis-----------------------------------------//
double I_sat;
double T_res[NMAX];
double w_t[NMAX], theta_t[NMAX];

//----2. Sensor-------------------------------------------------//
//2.1. Gyro
double r[NMAX]; //random walk
double w_m[NMAX];
```

```
//Gyro Noise
double w1=0.1; //w_noise
double w2=0.1; //noise random walk_dot

//2.2. Star data
double theta_m[NMAX];
//star noise
double v =0.01;

//----3. Kalman-----------------------------------------------------//
//matrixs
double A[MAX][MAX], B[MAX][MAX], Q[MAX][MAX];
//state equation dx/dt=Ax+Bw  Q=E[x]
double P_dot[MAX][MAX]; double PK[NMAX][MAX][MAX]; //covarian matrix
double KK[NMAX][MAX]; //Kalman Gain
double R; //=v*v //Measurement Noise

//system estimation
double theta_s[NMAX];       //estimation of theta angle
double theta_s_dot[NMAX];   //estimation of theta_dot
double r_s[NMAX];           //estimation of r

//----4. Controller-------------------------------------------------//
double K_p = -2; double K_d = -0.5;
double T_target[NMAX];

//----5. Reaction Wheel---------------------------------------------//
double I_rw =0.1;
double K_rw = 100;
double V_dot[NMAX],   V[NMAX];

//----Init values---------------------------------------------------//
//init Sat Axis
void init_Sat(){
      I_sat = 2.0;
      T_res[0] = 0;
      w_t[0]= 10*2*PI/60;
      theta_t[0] =50*PI/180;
}

//init sensor data
void init_Sensors(){
      //init Gyro data
      r[0]=0.03;
      w_m[0]= w_t[0] + genNoise(w1) + r[0];

      //init Star data
      theta_m[0] = 0; //before update time from Star
}
```

```
void init_RW(){
      //initial RW
      V_dot[0]=0;
      V[0]=0;
}

//---storeFiles-----------------------------------------------------//
void storeFile_NoKalman()
{
      FILE * fp;
      int  i;

      //store theta_t
      fp = fopen ("1.theta_t.txt", "w+");
      for(i=0;i<NMAX-1; i++)
      fprintf(fp,"%4.2f\n",theta_t[i]);
      fclose(fp);
      //store theta_m
      fp = fopen ("2.theta_m.txt", "w+");
      for(i=0;i<NMAX-1; i++)
      fprintf(fp,"%4.2f\n",theta_m[i]);
      fclose(fp);

      //store w_t
      fp = fopen ("3.w_t.txt", "w+");
      for(i=0;i<NMAX-1; i++)      fprintf(fp,"%4.2f\n",w_t[i]);
      fclose(fp);

      //store w_m
      fp = fopen ("4.w_m.txt", "w+");
      for(i=0;i<NMAX-1; i++)      fprintf(fp,"%4.2f\n",w_m[i]);
      fclose(fp);

      //store random walk
      fp = fopen ("5.r.txt", "w+");
      for(i=0;i<NMAX-1; i++)      fprintf(fp,"%4.3f\n",r[i]);
      fclose(fp);
}

//----Kalman Calculation-------------------------------------------//
//init for system estimation model
void init_kalman_matrix()
{
      //init A
      A[0][0]=0.0;          A[0][1]=1.0;
      A[0][2]=-1.0;
      A[1][0]=K_p/I_sat;    A[1][1]=K_d/I_sat;          A[1][2]=0.0;
      A[2][0]=0.0;          A[2][1]=0.0;
      A[2][2]=0.0;
```

```
	//init B
	B[0][0]=1.0;	B[0][1]=0.0;		B[0][2]=0.0;
	B[1][0]=0.0;	B[1][1]=1.0;		B[1][2]=0.0;
	B[2][0]=0.0;	B[2][1]=0.0;		B[2][2]=1.0;

	//init Q
	Q[0][0]=0.0;	Q[0][1]=0.0;		Q[0][2]=0.0;
	Q[1][0]=0.0;	Q[1][1]=w1*w1;		Q[1][2]=0.0;
	Q[2][0]=0.0;	Q[2][1]=0.0;		Q[2][2]=w2*w2;

	//init covarian matrix PK
	PK[0][0][0]=1.0; PK[0][0][1]=0.0;		PK[0][0][2]=0.0;
	PK[0][1][0]=0.0; PK[0][1][1]=1.0;		PK[0][1][2]=0.0;
	PK[0][2][0]=0.0; PK[0][2][1]=0.0;		PK[0][2][2]=1.0;
}

void init_kalman_estimation()
{
	//init  estimation
	r_s[0]=0;
	theta_s[0]=0;
	theta_s_dot[0]=0;
}

void storeFile_withKalman()
{
	FILE * fp;
	int  i;

	//store theta_s
	fp = fopen ("5.theta_s.txt", "w+");
	for(i=0;i<NMAX-1; i++)
	fprintf(fp,"%4.2f\n",theta_s[i]);
	fclose(fp);

	//store w_s
	fp = fopen ("6.w_s.txt", "w+");
	for(i=0;i<NMAX-1; i++)
	fprintf(fp,"%4.2f\n",theta_s_dot[i]);
	fclose(fp);

	//store random walk estimate
	fp = fopen ("7.r_s.txt", "w+");
	for(i=0;i<NMAX-1; i++)	fprintf(fp,"%4.3f\n",r_s[i]);
	fclose(fp);

	//PK
	fp = fopen ("8.p11.txt", "w+");
```

```
	for(i=0;i<NMAX-1; i++)
	fprintf(fp,"%4.3f\n",PK[i][0][0]);
	fclose(fp);

	fp = fopen ("9.p22.txt", "w+");
	for(i=0;i<NMAX-1; i++)
	fprintf(fp,"%4.3f\n",PK[i][1][1]);
	fclose(fp);

	fp = fopen ("10.p33.txt", "w+");
	for(i=0;i<NMAX-1; i++)
	fprintf(fp,"%4.3f\n",PK[i][2][2]);
	fclose(fp);
}

void init_Kalman(){
	init_kalman_matrix();
	init_kalman_estimation();
}

//calc P_dot (k)
void calc_P_dot(int k)
{
	double AT[MAX][MAX];
	double BT[MAX][MAX];
	double B1[MAX][MAX], B2[MAX][MAX]; //temp matrix
	double C1[MAX][MAX], C2[MAX][MAX], C3[MAX][MAX]; //temp matrix


	matrix_multi_matrix(A,PK[k],C1); //A*P

	calc_trans_matrix(A, AT);
	matrix_multi_matrix(PK[k],AT, C2);    //P*AT

	matrix_add_matrix(C1,C2,C3);     //A*P + P*AT

	matrix_multi_matrix(B,Q,B1);     //B*Q
	calc_trans_matrix(B, BT);
	matrix_multi_matrix(B1,BT,B2);   //B*Q

	matrix_add_matrix(C3,B2,P_dot); //A*P + P*AT + B*Q*BT
}

//calc Covarian Matrix PK(k)
void calc_PK(int k){
	int i,j;
	for(i=0;i<MAX;i++)
		{
			for(j=0;j<MAX;j++)
```

```
                        PK[k][i][j] = PK[k-1][i][j] + dt*P_dot[i][j];
            }
}

//calc Kalman Gain
void calc_KK(int k)
{
      double temp;

      R =v*v;

      temp=1/(PK[k][0][0] +R);
      KK[k][0] = temp*PK[k][0][0];
      KK[k][1] = temp*PK[k][1][0];
      KK[k][2] = temp*PK[k][2][0];
}

//update Covarian Matrix
void update_PK(int k)
{
      int i,j;
      double P_temp[MAX][MAX]    ;
      //copy PK[k] -->P_temp
      for(i=0;i<MAX;i++)
            {
                  for(j=0;j<MAX;j++)
                        P_temp[i][j] = PK[k][i][j];
            }

      PK[k][0][0]= (1.0-KK[k][0])*P_temp[0][0];
      PK[k][0][1]= (1.0-KK[k][0])*P_temp[0][1];
      PK[k][0][2]= (1.0-KK[k][0])*P_temp[0][2];

      PK[k][1][0]= P_temp[1][0] - KK[k][1]*P_temp[0][0];
      PK[k][1][1]= P_temp[1][1] - KK[k][1]*P_temp[0][1];
      PK[k][1][2]= P_temp[1][2] - KK[k][1]*P_temp[0][2];

      PK[k][2][0]= P_temp[2][0] - KK[k][2]*P_temp[0][0];
      PK[k][2][1]= P_temp[2][1] - KK[k][2]*P_temp[0][1];
      PK[k][2][2]= P_temp[2][2] - KK[k][2]*P_temp[0][2];
}

void estimate_state(int k){

            //estimate r_s = constant
            r_s[k]=r_s[k-1];

            //estimate theta_dot
            theta_s_dot[k]= w_m[k]-r_s[k];
```

```
            //Estimation for theta_s
            theta_s[k]= theta_s[k-1] + dt*theta_s_dot[k];

            //caculate p_dot
            calc_P_dot(k-1);
            //calc covarian matrix PK
            calc_PK(k);

}
//propagation and update
void update_phase(int k)
{
      calc_KK(k);      //calc Kalman Gain
      update_PK(k);    //update Covarian Matrix

//update state vector
//update theta_s_dot
theta_s_dot[k]= theta_s_dot[k] + (theta_m[k]-theta_s[k])*KK[k][1];

//update r_s
r_s[k] = r_s[k] + (theta_m[k]-theta_s[k])*KK[k][2];

//update theta_s_dot
theta_s[k] = theta_s[k] + (theta_m[k]-theta_s[k])*KK[k][0];
}
//---End of Kalman Calculation-----------------------------------//

//----Calculation System Cycle-----------------------------------//
void calc_requiredTorque(int k){

      //T_target[k]=  K_p*theta_m[k-1] + K_d*w_m[k-1]; //No KF
      T_target[k]=    K_p*theta_s[k-1] + K_d*theta_s_dot[k-1];
      //KF use theta_s  w_s
}

void calc_appliedTorque(int k){
            V_dot[k]   =     (K_rw/I_rw)*T_target[k];
            V[k]       =     V[k-1] + dt*V_dot[k];
            T_res[k]   =     (I_rw/K_rw)*V_dot[k];
}

void get_data_from_Sensor(int k){
      //true value, can not know
      w_t[k]           =     w_t[k-1] + dt*T_res[k]/I_sat;

      //true value
      theta_t[k]       =     theta_t[k-1] + dt*w_t[k-1];

      //simulate data , get real data from sensor if not simulation
```

```
	r[k]	=	r[k-1]	+ dt*genNoise(w2);
	//from gyro
	w_m[k]		=	w_t[k]	+ genNoise(w1)	+ r[k];

	//measurement of star, use for kf
	if(k%100 ==0 )
	{
		theta_m[k] =	theta_t[k] + genNoise(v);
	}
		else theta_m[k]=theta_m[k-1];
}

void system_cycle()
{
	int k; //steps of iteration
	for(k=1; k<NMAX; k++)
	{
		calc_requiredTorque(k); //from w and theta from KF
		calc_appliedTorque(k);
		get_data_from_Sensor(k);
		//estimate from kalman
		estimate_state(k);
		if(k%100==0) update_phase(k);
	}
}

void init_system(){

	//init random
	srand (1);
	init_Sat();
	init_Sensors();
	init_RW();

	init_Kalman();

}

int main()
{
	init_system();
	system_cycle();

	storeFile_NoKalman();
	storeFile_withKalman();
	return 0;
}
```