Computer Engineering
Mekelweg 4,
2628 CD Delft
The Netherlands
http://ce.et.tudelft.nl/

2019

MSc THESIS

IMPLEMENTATION AND VERIFICATION OF ENERGY EFFICIENT SOFTWARE FOR ADCS

HS Jagadeeshwar

Abstract

The Delfi-n3Xt was the second nanosatellite developed at the Delft University of Technology, and launched in 2013. Its successor Delfi-PQ is expected to be launched in first half of 2019. The attitude of a satellite can be referred to as its orientation in space with respect to a inertial reference frame. The Delfi-n3Xt was the first satellite from Delft University of Technology, to include three-axis Attitude Determination and Control System/Subsystem (ADCS). It was designed with 5 modes of operation. Four of these were advanced modes. In addition, the Delfi-PQ is not intended to include advanced modes of operation. Hence, this thesis considers using the Delfi-n3Xt ADCS software. This software is extended as a baseline implementation on the MSP-EXP432E401Y launchpad. Nearly, 32% of the total nominal power is assigned to ADCS. Hence, energy efficient design alternatives could be considered for future satellite missions. In addition, ADCS is a critical subsystem, failure of ADCS means failure of satellite mission. This thesis aims to improve performance and energy consumption of ADCS. This thesis considers study of three different Digital Signal Processing (DSP) alternatives: Double Precision (DP), Single Precision (SP) and Fixed Point (FxP) arithmetic. Study in this thesis concludes that FxP alternative provides ≈ 6.7 times better performance, and ≈ 7 times better energy efficiency over the baseline. Hence, this thesis proposes the use of FxP DSP alternative. It was also concluded that, the SP arithmetic has equivalent accuracy compared with DP. Moreover, SP provides ≈ 3 times better performance, and ≈ 2.7 times better energy efficiency over the baseline. Therefore, future implementations could benefit from an SP alternative. A major part of the ADCS power is allocated to sensor and actuator. This leaves only 10 % of the total nominal power assigned to ADCS software. Hence, the proposed alternative might not provide considerable improvements on total nominal power. However, for future satellite missions, if there exists a computationally intensive algorithm assigned with more significant amount of total nominal power, then, the proposed alternative could serve as an initial study. However, this does not guarantee that the suggested alternative could satisfy more accurate requirements. In such case, FxP implementation might result in accuracy violation. And use of SP alternative is proposed. In such case, a new study is suggested in order to benefit from FxP alternative.

CE-MS-2018-30

TU Delft
Delft University of Technology

Faculty of Electrical Engineering, Mathematics and Computer Science

IMPLEMENTATION AND VERIFICATION OF ENERGY EFFICIENT SOFTWARE FOR ADCS

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

HS Jagadeeshwar
born in Bellary, India

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

IMPLEMENTATION AND VERIFICATION OF ENERGY EFFICIENT SOFTWARE FOR ADCS

by HS Jagadeeshwar

Abstract

The Delfi-n3Xt was the second nanosatellite developed at the Delft University of Technology, and launched in 2013. Its successor Delfi-PQ is expected to be launched in first half of 2019. The attitude of a satellite can be referred to as its orientation in space with respect to a inertial reference frame. The Delfi-n3Xt was the first satellite from Delft University of Technology, to include three-axis Attitude Determination and Control System/Subsystem (ADCS). It was designed with 5 modes of operation. Four of these were advanced modes. In addition, the Delfi-PQ is not intended to include advanced modes of operation. Hence, this thesis considers using the Delfi-n3Xt ADCS software. This software is extended as a baseline implementation on the MSP-EXP432E401Y launchpad. Nearly, 32% of the total nominal power is assigned to ADCS. Hence, energy efficient design alternatives could be considered for future satellite missions. In addition, ADCS is a critical subsystem, failure of ADCS means failure of satellite mission. This thesis aims to improve performance and energy consumption of ADCS. This thesis considers study of three different Digital Signal Processing (DSP) alternatives: Double Precision (DP), Single Precision (SP) and Fixed Point (FxP) arithmetic. Study in this thesis concludes that FxP alternative provides ≈ 6.7 times better performance, and ≈ 7 times better energy efficiency over the baseline. Hence, this thesis proposes the use of FxP DSP alternative. It was also concluded that, the SP arithmetic has equivalent accuracy compared with DP. Moreover, SP provides ≈ 3 times better performance, and ≈ 2.7 times better energy efficiency over the baseline. Therefore, future implementations could benefit from an SP alternative. A major part of the ADCS power is allocated to sensor and actuator. This leaves only 10 % of the total nominal power assigned to ADCS software. Hence, the proposed alternative might not provide considerable improvements on total nominal power. However, for future satellite missions, if there exists a computationally intensive algorithm assigned with more significant amount of total nominal power, then, the proposed alternative could serve as an initial study. However, this does not guarantee that the suggested alternative could satisfy more accurate requirements. In such case, FxP implementation might result in accuracy violation. And use of SP alternative is proposed. In such case, a new study is suggested in order to benefit from FxP alternative.

Laboratory : Computer Engineering
Codenumber : CE-MS-2018-30

Committee Members :

Advisor:	Dr.ir Róbert Fónod, SSE, TU Delft
Chairperson:	Dr.ir Stephan Wong, CE, TU Delft
Member:	Prof. Dr. K.G. Langendoen, ES, TU Delft
Member:	Dr.ir Arjan van Genderen, CE, TU Delft

To the smile of my brother Prashanth & yet to be born
Nephew/Niece
Also to the hug of my friends Luc & Jiska

Contents

List of Figures

List of Tables

List of Acronyms

ADCS Attitude Determination and Control System/Subsystem

AEKF Additive Extended Kalman Filter

CORDIC COordinate Rotation DIgital Computer

CST Control Step Time

CT Compute Time

CP Compute Power

CSHT Control Step Hibernation Time

CSCT Control Step Compute Time

DP Double Precision

DSP Digital Signal Processing

DCM Direction Cosine Matrix

EPS Electrical Power System

EKF Extended Kalman Filter

ECI Earth Centered Inertial

ECEF Earth Centered Earth Fixed

FxP Fixed Point

FSP Fine Sun Pointing

FPGA Field-Programmable Gate Array

FPU Floating Point Unit

FWL Fraction Word Length

HAL Hardware Abstraction Layer

HP Hibernation power

HT Hibernation Time

IGRF International Geomagnetic Reference Field

IWL Integer Word Length

KF Kalman Filter

kB kiloByte

OBC On Board Computing

ORF Orbit Reference Frame

PD Proportional and Derivative

PLKF Pseudo Linear Kalman Filter

QS Quantisation steps

QKF Quaternion Kalman Filter

RTD Real Time Deadline

RQKF Reduced Quaternion Kalman Filter

SP Single Precision

SBF Satellites Body Fixed

TCT Total Compute Time

TCS Total Control Steps

WL Word Length

Acknowledgements

I thank my parents for their patience while I was taking my own time to complete my masters. With all the uncertainties they always showed their trust in me which boosted my self confidence.

I thank my supervisor Dr. ir. Stephan Wong for being a friendly and resourceful supervisor. In addition, he showed social concern and always motivated me. I also thank my co-supervisor Dr. ir. Róbert Fónod for being open-minded, transparent and helpful. When it was difficult for me to understand, a part of MATLAB code through mail, he invited me to a Skype call. And when I had a software bug for weeks, he sat next to me for many hours to help me understand and fix the bug. Most importantly, he understood the fact that I couldn't master understanding in Satellite Subsystems with my major in Embedded Systems. I also appreciate both of them for their help during my thesis writing.

I thank my academic counsellor Leonie Boortman and student counsellor John Stals, for their guidance during my entire master study.

I thank my friends Ayush, Pranav and Jonathan for involvement in deep brain stimulation. They always pointed out my determination even at my failures. Ayush was there even when I was stuck with a software bug late at night, he didn't completely understand my problem but he tried to help me. I also thank Ron and Marja for proof reading part of this thesis.

I take deep gratitude to thank my virtual family Luc, Jiska and Luc's parents. They were not part of the technical support club. But, they were there whenever I needed there help and always, supported and motivated me.

Lastly, I thank people who were directly or indirectly involved during my work that are not mentioned in this document.

HS Jagadeeshwar
Delft, The Netherlands
August 20, 2019

1 Introduction

The Delfi-PQ is third satellite project from Delft University of Technology. It focuses on the next level of satellite miniaturization. It is eight times smaller than its predecessors the Delfi-C3 and the Delfi-n3Xt. This also implies significant constraints on orbit average power consumption of 1 Watt [1]. The Delfi-n3Xt was launched in 2013 and its predecessor Delfi-C3 was launched in 2008. The Delfi-PQ will be developed iteratively and is expected to be launched in 2019.

Taking into consideration the increasing need for reduction in average power for future satellite projects it becomes prominent to focus on reduction in energy requirements. Hence, this thesis emphasizes implementation techniques to reduce energy consumption. In particular, it focuses on implementation and verification of energy efficient software for Attitude Determination and Control System/Subsystem (ADCS).

Attitude of a satellite is its orientation in space with respect to an inertial reference frame. ADCS, as the name indicates is a subsystem that determines the attitude of the satellite and controls the actuators in order to reach a desired orientation in space. As the future missions move towards new advancements, there is increasing focus, towards the precision requirements. For example, more precise imagery of the Earth, Sun pointing for more efficient energy harvesting, thruster pointing for orbit maneuvers, antenna pointing for communication with the ground station and many more applications [5]. For proper functioning of the Delfi-n3Xt satellite, it was critical to have the rotational rates below 1 °/s. And this required an active ADCS [1]. During the early operation of the Delfi-n3Xt, the satellite was spinning up increasing the rotational rate from 5 °/s to 45 °/s the next day. Thankfully due to the presence of parameter upload functionality, it was possible to switch the actuation direction of two of the magnetorquers [6]. In the worst case, failure in ADCS also means failure in the satellite mission, as the rotational rate can increase beyond operational limits. Therefore, robust and reliable ADCS is critical for proper functioning of satellites also in upcoming missions. It is worth noting that energy saving in ADCS means more energy available for other critical subsystems, such as On Board Computing (OBC).

1.1 Previous work

In addition to satellites launched by the Delft University of Technology, there are several other nano/pico satellites equipped with an ADCS. Database list in [2] highlights several other satellite projects along with mission description like, TUBSAT-N, TUBSAT-N1, Artemis JAK, Artemis Louise, Artemis Thelma, ASUSAT-1, etc. There were several

[1] https://www.tudelft.nl/lr/delfi-space/delfi-pq

[2] https://www.nanosats.eu/

master thesis works done at Delft University of Technology, which dealt with requirements, design, implementation, and verification of ADCS algorithm [7][8][5][1].

Using spacecraft miniaturization CubeSats have proven their feasibility of low cost and short development time by use of commercial-of-the-self components. The reference document [9] presents a comparison between CubeSats and PocketQubes, and analyzes the impact of miniaturization on spacecraft design and performance. It also presents the preliminary design of the Delfi-PQ and concludes with a development philosophy. Satellite miniaturization in other terms means the reduction in satellite volume. This reduction in volume not only limits the available antenna size but also impacts the available power, which in turn make energy efficiency and management more critical.

A new architecture for satellite electrical subsystem in order to reduce volume and increase the empty surface usage is presented in [10]. It discusses a modular approach by splitting the electrical power system on different surfaces and reducing the number of voltage regulators required. It provides solutions that reduce the number of cables running in satellite, which makes integration simpler. Their research goals focus on turning the Electrical Power System (EPS) into a more flexible, scalable and volume-efficient system by physically relocating its components and a lean approach.

The experiences and lessons learned from other missions are of great importance for future satellite missions. The paper in-orbit results of the Delfi-n3Xt: Lessons learned and move forward, presents in orbit results of the Delfi-n3Xt satellite [6]. They demonstrate the success of in-orbit payloads and platform. In particular, four of them, including a solid cool gas micro-propulsion system, a new type of solar cell, a more robust Command and Data Handling Subsystem (CDHS), and an integrated ADCS that perform three-axis active control using magnetorquers and reaction wheels. They demonstrate the following:

- Two cold gas generators and several thrust maneuvers are used to demonstrate ignition of a solid cool gas micro-propulsion system. The measurements of internal gas pressure and temperature are recorded at a frequency of 30Hz, that provide detailed information about the system performance.
- Use of a new type of material a-Si:H with higher radiation tolerance, better annealing properties, and a higher power-versus-weight ratio without using covering glass.
- More robust Command and Data Handling Subsystem (CDHS), which show fewer communication errors and bus lockups compared to Delfi-C3.
- ADCS that performs active three-axis control using magnetorquers and reaction wheels. Even with the presence of high noise in sensors and actuators ADCS proves to work correctly.

1.2 Problem statement

As the future satellite missions move towards miniaturization, reduction in volume has a tremendous impact on the available power and makes energy efficiency and management

more critical [9]. Failure to meet such energy requirements means the requirement for a larger battery for proper functioning of the satellite, which also means to fail in future miniaturization.

The only satellite mission launched with active three-axis ADCS from the Delft University of Technology is the Delfi-n3Xt. Most recent mission the Delfi-PQ, however, does not incorporate an advanced mode that involves intensive computation. Moreover, the source code for this mission is still in the testing phase. Hence, this thesis work uses the source code developed as part of the Delfi-n3Xt mission.

Study on CubeSats to pocketqubes opportunities and challenges suggests the use of MSP432P401R microcontroller, as the best-suited possibility among two other implementation platforms. This study [9] may no longer hold for more advanced and precise algorithms to be used in future missions. There is a need for a faster implementation platform running at much higher clock frequencies in order to meet Real Time Deadline (RTD). Flash memory of microcontroller becomes a crucial element in order to choose a microcontroller, as the requirement of more precise International Geomagnetic Reference Field (IGRF) look-up requires more memory (for example, IGRF look-up, using Single Precision (SP) data-storage, with 1 $^\circ$ of precision requires 759.375 kiloByte (kB) of memory). Microcontroller MSP432P401R with a flash memory of 256 kB provisions insufficient flash memory. Hence, this may no longer be used.

Requirements for ADCS for the Delfi-n3Xt has been studied in document [1]. It states that the rotation rate must be below 1 $^\circ$/s, and sun-pointing error must be below 25 degrees. This thesis directs on these two critical requirements. In order to meet this it is essential to have the following two modes of operation:

- **Detumble mode**: Once the satellite is ejected from launch container Detumbling mode must bring the satellite rotational rate down to 1 $^\circ$/s [8].
- **Fine Sun Pointing (FSP) mode**: This mode uses Extended Kalman Filter (EKF) algorithm using both Sun sensor & magnetometer measurements to point solar-panels towards the Sun. The acceptable sun-pointing error is a maximum of 25 $^\circ$ in a non-eclipse period.

1.3 Thesis objective

The objective of this thesis work is to implement and verify energy efficient software for the ADCS algorithm. Future satellite missions could benefit from the conclusions or use the software framework if relevant. The following research questions are formulated to accomplish the thesis objective:

- How can we develop a software implementation from the Delfi-n3Xt ADCS algorithm which is more energy efficient?
- How could we verify its functionality and test if it meets the accuracy requirements?

From the above mentioned research questions the following two sub-questions are listed:

1. Is it necessary to use double-precision arithmetic to satisfy accuracy requirements for such a system?

2. Can we practically compare the energy efficiency of such a software implementation with its baseline the Delfi-n3Xt ADCS algorithm?

1.4 Research methodology

The implementation and verification of energy efficient software for ADCS can be divided in several stages:

Firstly, literature study is done. It includes study of following:

1. ADCS in general
2. ADCS in Delfi project
3. Kalman Filter (KF) in general
4. EKF in Delfi-n3Xt
5. Precision requirements
6. Accuracy requirements

Secondly, the simulation environment (in MATLAB) developed for the Delfi-n3Xt is modified. Especially, two main modifications are considered:

1. Modes of operation other than FSP and detumble mode are removed for simplicity. As can be seen in Figure 1.2 modes encircled in red (Coarse Sun Pointing mode, Ground Station tracking mode and Thruster Pointing mode) will be removed from the design.
2. To print sensor and actuator data into a text file which is used for open-loop C code design.

Thirdly, C code from the Delfi-n3Xt will be redesigned. Two main modifications are done:

1. The Delfi-n3Xt has a three layer software structure as shown in Figure 1.1. The Hardware Abstraction Layer (HAL) layer is modified in this thesis. In this layer the sensor/actuator read/write are modified by read/write from text files.
2. Similar to the simulation environment, for simplicity, modes of operation, other than, FSP and detumble mode are removed.
3. C based source code is built using CMake [3]. This is done on a MacBook-Pro running macOS Mojave operating system. Moreover, this simulation is an open-loop simulation using text files.

Fourthly, a complete implementation and testing of FSP mode with EKF and Proportional and Derivative (PD) controller is done. It is important to note that C

[3] https://cmake.org/

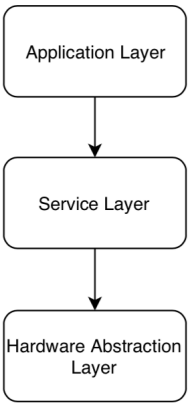


Figure 1.1: Layered software structure of ADCS in the Delfi-n3Xt

code written for the Delfi-n3Xt project although included FSP mode the implementation and verification was incomplete. Results from the previous, step open-loop C code based simulation are compared with closed-loop MATLAB simulation. And this is reiterated until acceptable accuracy requirement is satisfied. Three versions of this design are considered, which will act as:

1. Double Precision (DP): The baseline implementation.

2. SP: The first improved implementation.

3. Fixed Point (FxP): The second improved implementation.

Fifthly, MATLAB and open loop C-code are combined together to form a closed loop design. Text file read/write are replaced with sensor/ actuator data exchange using TCP/IP server/client.

Lastly, hardware in a loop simulation is done. Figure 1.3 shows closed loop simulation of ADCS with hardware in a loop. That i,s C code built using CMake is ported to TI-RTOS built on MSP432 launchpad (MSP-EXP432E401Y). Measurements of speed-up and energy for three versions of design are recorded. The kinematics and dynamics marked with gree inn Figure 1.3 includes properties of motion and the rules governing the interactions. Using these models the position, velocity, angular rate and attitude of the satellite are propagated in time [5].

1.5 Contributions

ADCS is a large software framework. Following are the contributions:

- A preexisting implementation from the Delfi-n3Xt project is modified as mentioned in the previous section. This DP baseline ADCS is extended with the implementation of FSP mode, which was incomplete in the Delfi-n3Xt project.

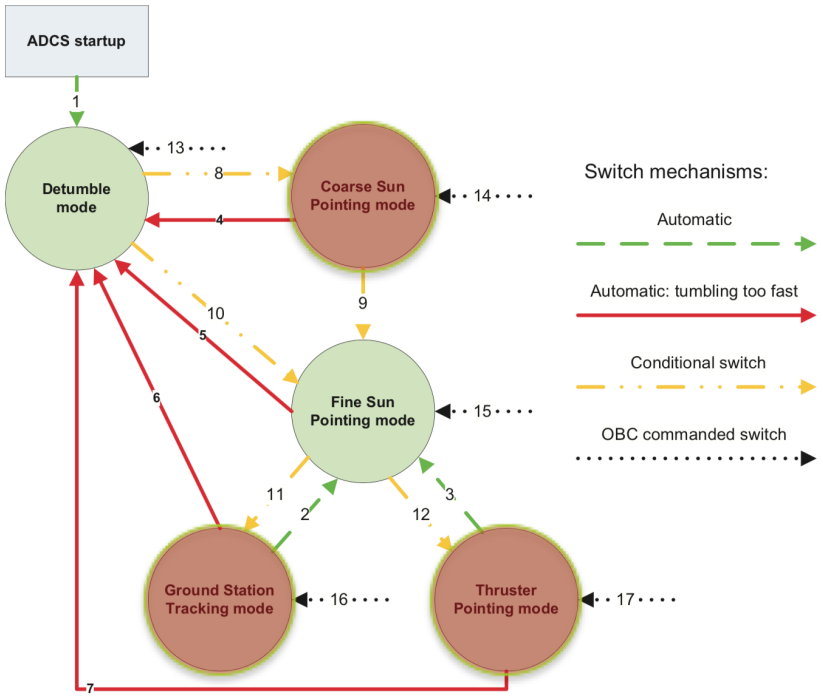


Figure 1.2: The possible mode switches of the ADCS [1]

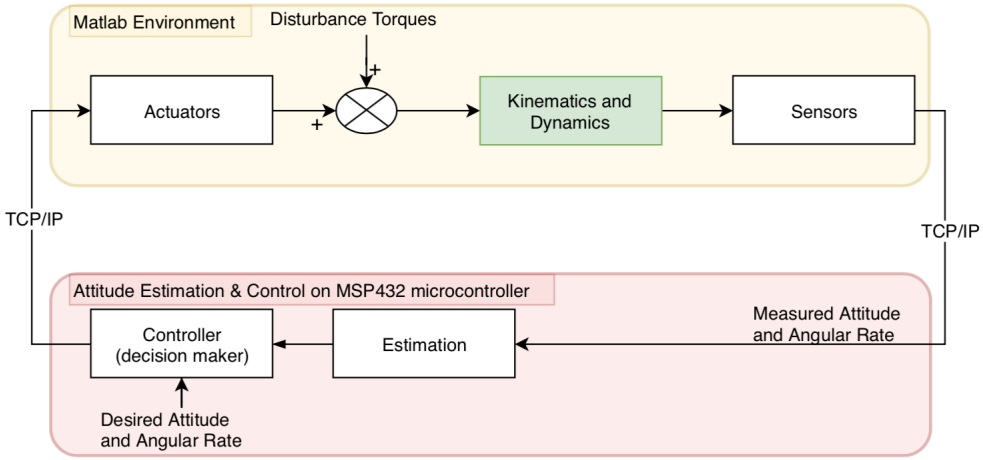


Figure 1.3: Simulation of ADCS with hardware in a loop

- This ADCS algorithm is ported completely to SP and FxP. It also includes implementing a SP and FxP matrix library.
- In order to test the functionality of the complete ADCS algorithm including sensor, actuator and environment model, a hardware in loop approach is used. This in-

cludes ADCS with detumble and FSP mode implemented in MSP-EXP432E401Y microcontroller. Where as sensor, actuator, and environmental model are implemented in MATLAB.

- A MATLAB script that generates IGRF look-up is written for both floating-point and FxP implementation.

1.6 Thesis outline

This document starts with an introductory chapter (current chapter). This chapter presents the research question, thesis objective, research methodology and personal contribution. Chapter 2 presents an introduction to ADCS, i.e., the blocks marked red in the Figure 1.3. This chapter starts with the definition of reference frames, followed by definition and description of attitude estimation and control. It also defines kinematics and dynamics of the satellite attitude. Chapter 3 zooms in to describe the computation used for ADCS. That is, the Digital Signal Processing (DSP) alternatives. It introduces the floating-point arithmetic followed by fixed-point arithmetic. It describes the three versions of design alternatives. Namely, DP, SP and FxP. Chapter 4 is about experimental results. More specifically, it includes simulation results along with platforms and methods used to acquire the results. Lastly, chapter 5 presents the summary, conclusions, main contributions, and future work.

Attitude Determination and Control System

2

The Attitude of a satellite can be referred to as its orientation in space with respect to inertial reference frame. Motion of a satellite in space can be specified by its position, velocity, orientation, and rotational rate. Attitude determination is the process of measuring and computing the orientation of satellite in its body frame with respect to a reference frame. This chapter begins with defining reference frames. Secondly, rotational kinematics is presented, which provide a description for vector transformations. In particular, three attitude representations. Thirdly, it describes model of physical world. This is required in order to calculate the position, and velocity of the satellite, the disturbance torques, the Sun direction, the local Earth magnetic field, and the ground station direction respectively. Moreover, in order to estimate the attitude of satellite it is essential to use estimation algorithm such as, EKF. The Delfi-n3Xt uses EKF for attitude estimation. Fourthly, this chapter defines attitude estimation and describes in detail EKF algorithm. Attitude control is the process of correcting the orientation of satellite in a specified direction. This correction is achieved by controlling the actuators of the satellite. Fifthly, this chapter describes attitude control algorithms that are used in the Delfi-n3Xt. The attitude kinematics and dynamics of the satellite are modelled in MATLAB. Hence, this chapter will not describe this, for more detailed explanation, readers are referred to [1]. Lastly, this chapter describes power budget for ADCS in the Delfi-n3Xt.

Objective of ADCS in this thesis work are listed below:

- To reduce angular velocity of the satellite below 1 $^{\circ}$/s in detumble mode.
- To point solar panels towards the Sun, in order to maximize energy harvesting. Maximum allowed Sun pointing error is 25 $^{\circ}$ in FSP mode.

2.1 Reference frames

ADCS uses various reference frames to represent the orientation of the satellite. A reference frame can be described as a Cartesian coordinate system with a set of three orthogonal unit vectors. Rotation matrix can be used to represent attitude between two reference frames. The below described reference frames are used to describe orientation between different objects. Description of methods used to represent the orientation of these frames with respect to the Earth Centered Inertial (ECI) reference frame, and methods used to represent the orientation of these frames with respect to the Satellites Body Fixed (SBF) reference frame, are not described in this document and readers are referred to [1].

2.1.1 ECI reference frame

ECI reference frame Figure 2.1 has origin fixed at center of mass of the Earth. The z-axis is perpendicular to the x-axis and extends through the celestial North pole, x-axis lies in equatorial plane and points to vernal equinox on the celestial sphere, and the y-axis is perpendicular to both x and z-axis and completes the right-handed reference frame [8]. ECI reference frame is referred as a unit vector I = $\{\vec{i}_1,\vec{i}_3,\vec{i}_3\}$.

2.1.2 Earth Centered Earth Fixed (ECEF) reference frame

ECEF reference frame Figure 2.1 [1] is a rotating frame fixed to the Earth. The z-axis is pointing north parallel to the rotation axis of the Earth, x-axis of the ECEF reference frame is pointing towards 0° latitude and and 0 ° longitude, and the y-axis is perpendicular to both x and z-axis and completes the right-handed reference frame [8]. ECEF reference frame is referred as a unit vector E = $\{\vec{e}_1,\vec{e}_2,\vec{e}_3\}$.

The main difference of this reference frame with respect to ECI is, the ECEF reference frame rotates along with the Earth. Consequently, a point fixed on the Earth surface has fixed coordinates and do not change with respect to the ECEF reference frame.

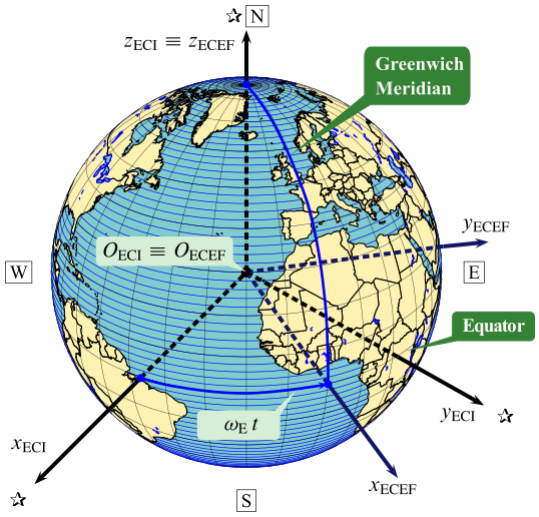


Figure 2.1: The ECI and ECEF reference frame

2.1.3 Satellite Orbit Reference Frame (ORF)

The Satellite ORF has origin fixed at the center of mass of the satellite. The z-axis of the ORF points towards the Nadir (the direction of the center of the Earth), y-axis points toward orbital plane or tangent line of the orbit (i.e., the velocity vector), and the x-axis is perpendicular to both y and z-axis and completes the right-handed reference frame [8]. ORF is referred as a unit vector O = $\{\vec{o}_1,\vec{o}_2,\vec{o}_3\}$.

[1] https://www.picswe.com/pics/centered-inertial-31.html

2.1.4 SBF reference frame

The SBF reference frame has its origin at the center of mass of satellite. All the three axes are aligned with the three principle axes of inertia of the satellite [8]. ORF is referred as a unit vector B = $\{\vec{b}_1,\vec{b}_2,\vec{b}_3\}$.

2.1.5 The Sun pointing reference frame

The Sun pointing reference frame has its origin at the centre of mass of the satellite. The z-axis points towards Sun, the y-axis points towards the projection of the inertial frame on the plane perpendicular to the Sun direction, the x-axis is perpendicular to both y and z-axis and completes the right-handed reference frame [1]. The rotation around the Sun direction is free. Hence, an infinite number of Sun pointing reference frames can be defined. The specific frame chosen here requires the calculation of only two angles, which are not discussed in this document refer [1] for more details.

2.2 Rotational kinematics

This section will provide a description of vector transformation between different reference frames. And, the orientation of a reference frame with respect to another reference frame. This section will also provide a description of orientation when one of the reference frames is in rotational motion, with respect to the other reference frame. There are three prominent ways in which the attitude of satellites can be represented. Namely Direction Cosine Matrix (DCM), Euler angles, and the Quaternion representation. The Delfi-n3Xt uses all these three representations. The equations described below are used to propagate the attitude of satellite.

2.2.1 Direction Cosine Matrix representation

Consider ECI reference frame I, and SBF reference frames B, with a set of right-handed orthogonal unit vectors $\{\vec{i}_1, \vec{i}_2, \vec{i}_3\}$ and $\{\vec{b}_1, \vec{b}_2, \vec{b}_3\}$ [1]. Basis vectors of I can be expressed in terms of B by following equations [2]:

$$\vec{i}_1 = C_{13}\vec{b}_3 + C_{12}\vec{b}_2 + C_{11}\vec{b}_1 \tag{2.1a}$$

$$\vec{i}_2 = C_{23}\vec{b}_3 + C_{22}\vec{b}_2 + C_{21}\vec{b}_1 \tag{2.1b}$$

$$\vec{i}_3 = C_{33}\vec{b}_3 + C_{32}\vec{b}_2 + C_{31}\vec{b}_1 \tag{2.1c}$$

$$\vec{i}_1.\vec{b}_1 = C_{11} \quad (2.2a) \qquad \vec{i}_2.\vec{b}_1 = C_{21} \quad (2.2b) \qquad \vec{i}_3.\vec{b}_1 = C_{31} \quad (2.2c)$$

$$\vec{i}_1.\vec{b}_2 = C_{12} \quad (2.3a) \qquad \vec{i}_2.\vec{b}_2 = C_{22} \quad (2.3b) \qquad \vec{i}_3.\vec{b}_2 = C_{32} \quad (2.3c)$$

$$\vec{i}_1.\vec{b}_3 = C_{13} \quad (2.4a) \qquad \vec{i}_2.\vec{b}_3 = C_{23} \quad (2.4b) \qquad \vec{i}_3.\vec{b}_3 = C_{33} \quad (2.4c)$$

Direction cosine, $\boldsymbol{C}_{kl} = \vec{i}_k.\vec{b}_l$ is the cosine of the angle between $\vec{i}_k$ and $\vec{b}_l$. In other words, relation can be written as:

$$\begin{bmatrix} \vec{i}_1 \\ \vec{i}_2 \\ \vec{i}_3 \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \begin{bmatrix} \vec{b}_1 \\ \vec{b}_2 \\ \vec{b}_3 \end{bmatrix} = \boldsymbol{C}^{I/B} \begin{bmatrix} \vec{b}_1 \\ \vec{b}_2 \\ \vec{b}_3 \end{bmatrix} \tag{2.5}$$

Where $\boldsymbol{C}^{I/B}$ is called as DCM rotation matrix or coordinate transformation matrix to I from B. Since, both reference frames B and I consists of orthogonal unit vectors, both the DCM's are orthonormal matrix, we have an important relationship, defined below:

$$\left[\boldsymbol{C}^{I/B}\right]^{-1} = \left[\boldsymbol{C}^{I/B}\right]^{T} = \boldsymbol{C}^{B/I} \tag{2.6}$$

Hence, if we have DCM $C^{I/B}$ then we can easily transform to $C^{B/I}$, and vice-versa. Another important relationship is:

$$\boldsymbol{C}^{C/B} = \boldsymbol{C}^{C/I}\boldsymbol{C}^{I/B} \tag{2.7}$$

Hence, if we want to transform a vector from reference frame B to C i.e., DCM $\boldsymbol{C}^{C/B}$ we can do so by two successive rotations $\boldsymbol{C}^{C/I}$ and $\boldsymbol{C}^{I/B}$.

2.2.2 Euler angle representation

As we can see from previous subsection DCM requires 9 parameters to describe attitude of a satellite, out of which 6 are redundant. DCM can be used to perform transformation from one reference frame to other. The orientation of frame I relative to frame B can also be described by three successive rotations around the axis of the SBF reference frame B. For example, consider a rotation angle of ψ on z axis, θ on y axis and ϕ on x axis, also referred as roll, pitch and yaw as shown in Figure 2.2.

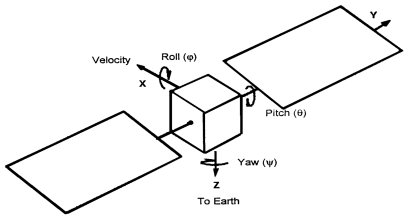


Figure 2.2: Satellite Coordinate Frame Definition [2]

There are in total 27 rotation sequences possible out of which only 12 hold the case that no two consecutive rotations sequences can be same. These valid rotations are: $(i, j, k) \in$
$\{(3,1,2),(3,1,3),(3,2,1),(3,2,3),(1,2,1),(1,2,3),(1,3,1),(1,3,2),$
$(2,1,2),(2,1,3),(2,3,1),(2,3,2)\}$. In other words, successive rotation $(1,2,3)$ can be expressed by equation 2.8. Figure 2.3 represents this rotational sequence pictorially.

$$\boldsymbol{C}C^{B/A} = \boldsymbol{C}_{123}(\phi,\theta,\psi) = \boldsymbol{C}_1(\phi)\boldsymbol{C}_2(\theta)\boldsymbol{C}_3(\psi) \tag{2.8}$$

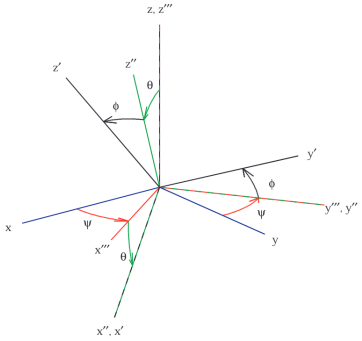


Figure 2.3: Euler angle sequence (1,2,3) [3]

This can be expressed as following DCM's [3]

$$\boldsymbol{C}_1(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \tag{2.9}$$

$$\boldsymbol{C}_2(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{2.10}$$

$$\boldsymbol{C}_3(\psi) = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.11}$$

By substituting equation 2.9,2.10, and 2.11 in equation 2.8 and by replacing sin with s and cos with c we get:

$$\boldsymbol{C}^{I/B} = \begin{bmatrix} \hat{i}_1 \\ \hat{i}_2 \\ \hat{i}_3 \end{bmatrix} = \begin{bmatrix} c_\theta c_\psi & c_\theta s_\psi & -s_\theta \\ s_\phi s_\theta c_\psi - c_\phi s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\theta s_\phi \\ c_\phi s_\theta c_\psi + s_\phi s_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi & c_\theta c_\phi \end{bmatrix} \begin{bmatrix} \hat{b}_1 \\ \hat{b}_2 \\ \hat{b}_3 \end{bmatrix} \tag{2.12}$$

Although it is much less computationally intensive to use 3 variable Euler angles they are said to suffer from singularity that arise from *gimble lock* [3]. Intuitively, *gimble lock* is said to arise when there is indistinguishably of changes in the first and third Euler angles when the second Euler angle is at some critical value. For example, for a rotational sequence $(1, 2, 3)$, when $\theta = 90^\circ$ (Pitch) the satellite is pointing straight up, and ψ (roll) and ϕ (yaw) are indistinguishable.

2.2.3 Quaternion representation

Before we can define quaternions it is essential to understand Euler's eigenaxis rotation theorem. It states that [2]:

"The rigid body attitude can be changed from any given orientation to any other orientation by rotating a rigid body about an axis that is stationary in an inertial reference frame and fixed to the body reference frame. Such an axis of orientation, whose orientation relative to both inertial and the body reference frame remains unchanged throughout the motion, is called the Euler axis or eigenaxis." ***-Leonhard Euler***

Let us consider two reference frames B and I with a set of right-handed orthogonal unit vectors $\{\vec{b}_1, \vec{b}_2, \vec{b}_3\}$ and $\{\vec{i}_1, \vec{i}_2, \vec{i}_3\}$. The orientation of I with respect to B can be characterized by an unit vector $\vec{e}$ along the eigenaxis. The rotation around eigenaxis can be characterized by:

$$\vec{e} = \vec{e}_1\vec{b}_1 + \vec{e}_2\vec{b}_2 + \vec{e}_3\vec{b}_3 = \vec{e}_1\vec{i}_1 + \vec{e}_2\vec{i}_2 + \vec{e}_3\vec{i}_3 \tag{2.13}$$

Where $\vec{e}_1$,$\vec{e}_2$, and $\vec{e}_3$ are the direction cosines of the eigenaxis relative to both reference frames B and I. The Euler axis or eigenaxis rotation is shown in Figure 2.4.

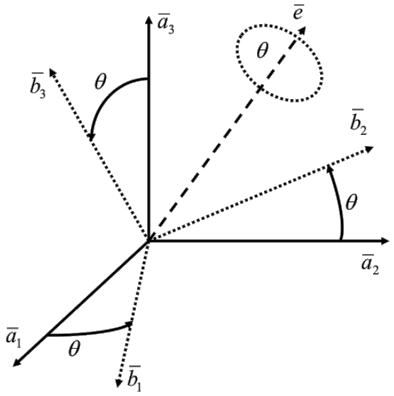


Figure 2.4: Euler axis or eigenaxis rotation

Quaternions are one of the most powerful way of attitude representation. They use the above stated Euler's eigenaxis rotation theorem. Quaternions can be represented by 4 parameters, of which one represents scalar part and the rest represent vector part. This can be depicted by:

$$\boldsymbol{q} = \begin{bmatrix} \boldsymbol{q} \\ q_4 \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \tag{2.14}$$

Where, components q_1, q_2, and q_3 represents vector part, and q_4 represents scalar part. These components can be rewritten in terms of eigenaxis $\hat{\boldsymbol{e}}$ as:

$$q_1 = \hat{e}_1 \sin\left(\frac{\theta}{2}\right) \tag{2.15a}$$

$$q_2 = \hat{e}_2 \sin\left(\frac{\theta}{2}\right) \tag{2.15b}$$

$$q_3 = \hat{e}_3 \sin\left(\frac{\theta}{2}\right) \tag{2.15c}$$

$$q_4 = \cos\left(\frac{\theta}{2}\right) \tag{2.15d}$$

These 4 components must satisfy the below equation:

$$\sqrt{{q_1}^2 + {q_2}^2 + {q_3}^2 + {q_4}^2} = \|\boldsymbol{q}\| = 1 \tag{2.16}$$

Just as DCM $\boldsymbol{C}^{B/I}$ can represent attitude of satellite the same way quaternion $\boldsymbol{q}^{B/I}$ can also do so. In other words, DCM and quaternion can be also be represented in forms of each other. Equation below describes DCM $C^{B/I}$ in terms of quaternion $\boldsymbol{q}^{B/I}$:

$$\boldsymbol{C}^{B/I}(\boldsymbol{q}^{B/I}) = \begin{bmatrix} {q_1}^2 - {q_2}^2 - {q_3}^2 + {q_4}^2 & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) \\ 2(q_2q_1 - q_3q_4) & 1 - 2({q_1}^2 + {q_3}^2) & 2(q_2q_3 + q_1q_4) \\ 2(q_3q_1 + q_2q_4) & 2(q_3q_3 - q_1q_4) & 1 - 2({q_1}^2 + {q_2}^2) \end{bmatrix} \tag{2.17}$$

$$= ({q_4}^2 - \|\boldsymbol{q}\|^2)I_3 + 2\boldsymbol{q}\boldsymbol{q}^T - 2[\boldsymbol{q}\times] \tag{2.18}$$

Where, $\boldsymbol{q}$ is the vector part of quaternion, and $[\boldsymbol{q}\times]$ is the skew symmetric matrix defined by:

$$[\boldsymbol{q}\times] = \begin{bmatrix} 0 & -q_3 & q_2 \\ q_3 & 0 & -q_1 \\ -q_2 & q_1 & 0 \end{bmatrix} \tag{2.19}$$

Similarly, we can describe quaternion in terms of DCM as:

$$q_1 = \frac{1}{4q_4}(C_{23} - C_{32}) \tag{2.20a}$$

$$q_2 = \frac{1}{4q_4}(C_{31} - C_{13}) \tag{2.20b}$$

$$q_3 = \frac{1}{4q_4}(C_{12} - C_{21}) \tag{2.20c}$$

$$q_4 = \pm\frac{1}{2}\sqrt{(1 + C_{11} + C_{22} + C_{33})} \tag{2.20d}$$

Similar to DCM quaternion can also easily be transformed from for example $\boldsymbol{q}^{B/A}$ to $\boldsymbol{q}^{A/B}$ by taking a conjugate as shown by equation below:

$$\boldsymbol{q}^{A/B} = \bar{\boldsymbol{q}}^{B/A} \tag{2.21}$$

Where conjugate of quaternion $\bar{\boldsymbol{q}}^{B/A}$ is given by:

$$\bar{\boldsymbol{q}} = \begin{bmatrix} -\boldsymbol{q} \\ q_4 \end{bmatrix} = \begin{bmatrix} -q_1 \\ -q_2 \\ -q_3 \\ -q_4 \end{bmatrix} \tag{2.22}$$

Furthermore, rotation as in DCM given by equation 2.5 can be done in quaternions using a similar property. This can be expressed by a multiplication of quaternions that is shown by equation below:

$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ 0 \end{bmatrix} = \bar{\boldsymbol{q}}^{B/A} \otimes \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ 0 \end{bmatrix} \otimes \boldsymbol{q}^{B/A} \tag{2.23}$$

Where operator $\otimes$ stands for quaternion multiplication. Quaternion multiplication is very similar to complex number multiplication with a special consideration that order of multiplication is considered strictly and have to be followed in order. In other words, out of order multiplication of quaternion is said to be violation of quaternion multiplication rule and will lead to an error in result of multiplication. Lastly, frame transformation property of DCM as in equation 2.6 can also be achieved by quaternion using following equation:

$$\boldsymbol{q}^{C/A} = \boldsymbol{q}^{C/B} \otimes \boldsymbol{q}^{B/A} \tag{2.24}$$

Two main advantages of quaternion are:

1. Compared to DCM quaternion are very compact i.e., quaternion only require 4 parameters while DCM requires 9.

2. Singulatularity issues that were mentioned to arise from *gimble lock* in Euler angles are not present in quaternion. By using 4 parameters in quaternion instead of 3 in Euler angles we get rid of Singulatularity.

In the Delfi-n3Xt, quaternion representation are used in Quaternion Feedback Regulator within FSP mode. This will be described in section 2.5.

2.3 Model of the physical world

The physical model of the world is necessary to investigate the performance of satellite's ADCS. These models are necessary to calculate the position, and velocity of the satellite, the disturbance torques, the Sun direction, the local Earth magnetic field, and the ground station direction respectively. But, this thesis work only emphasizes the use of the local Earth magnetic field and the Sun direction, which are described in subsections below. For other physical models readers are referred to [1].

2.3.1 Magnetic field of the Earth

The International Association of Geomagnetism and Aeronomy (IAGA) recommends the use of IGRF for scientific work to empirically represent the Earth's magnetic field. The coefficients for the IGRF model are based on all available data sources including geomagnetic measurements from observatories, ships, and satellites. The standard mathematical description of the Earth's magnetic field of IGRF has its latest release in 2014. But, the Delfi-n3Xt was launched in 2013 and the latest version that was available at that time was from 2010. Hence, this thesis work employs the use of 2010 release.

Using the ECEF position of the satellite, IGRF can be used calculate the Earth's magnetic field. This is accomplished by using a series of mathematical models of the Earth's main field and its annual rate of change (secular variation). The source internal to the Earth is the negative gradient of a scalar potential V which can be represented using a truncated series expansion [2]:

$$V(r,\theta,\phi,t) = a\sum_{n=1}^{N}\sum_{m=0}^{n}\left(\frac{a}{r}\right)^{n+1}\left[g_n^m(t)\cos(m\phi) + h_n^m(t)\sin(m\phi)\right]P_n^m(\cos\theta) \quad (2.25)$$

In the Equation 2.25:

- r = radial distance from the center of the Earth
- a= geomagnetic conventional Earth's mean reference spherical radius = 6371.2 km
- θ = geocentric co-latitude
- ϕ = east longitude
- $P_n^m(\cos\theta)$ = Schmidt quasi-normalized, Legendre functions of degree n and order m
- $g_n^m(t)$ and $h_n^m(t)$ = main field (MF) at epochs separated by 5 years between 1900.0 and 2015.0 A.D
- t = time of interest in years

Detailed description on the mathematical computations, for Equation-2.25 can be found in [11].

Although the Delfi-n3Xt formulated use of IGRF this implementation was not done. There are two possibilities to use IGRF model. Firstly, implementing the complex computations on-board to directly calculate Earth's magnetic field using the position of the satellite in the ECEF frame as input. Secondly, generating a look-up table that can be used to look-up Earth's magnetic field using the position of the satellite in the ECEF frame as input. The second option is chosen for this thesis work as the on-board computation would require a considerable compute power.

Before we can use IGRF model, it is required to convert the ECEF position to spherical coordinates. Spherical coordinates (r,θ,ϕ), where r is radial distance, θ is polar/elevation angle, and ϕ is azimuthal angle. The radial distance is the distance of the point from the fixed origin. The elevation angle is angle measured from a fixed zenith direction (The zenith is an imaginary point directly above a particular location, on the imaginary celestial sphere [3]). The azimuth angle is orthogonal to the zenith and has orthogonal projection on a reference plane that passes through the origin and is measured from a fixed reference direction on that plane. This is shown in Figure 2.5. To be more specific IGRF uses geographic coordinate system. The geographic coordinate system uses latitude, longitude and elevation to enables every location on Earth Figure 2.6.

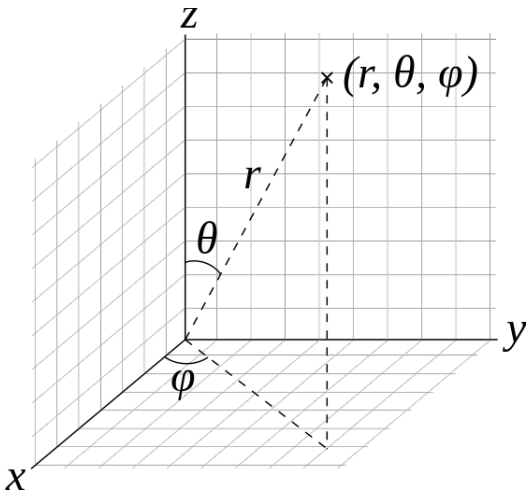


Figure 2.5: Spherical coordinates(r,θ,ϕ) [4]

This has a simple conversion from spherical coordinates, i.e., latitude = polar - 90°, and longitude = azimuthal.

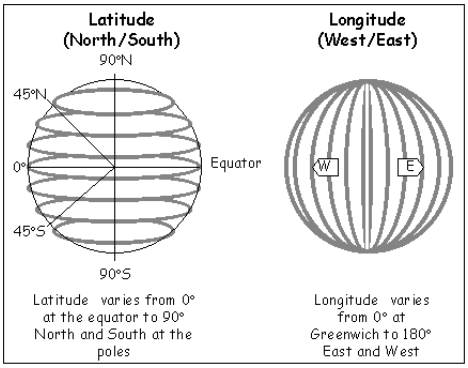


Figure 2.6: geographic coordinate system [5]

A MATLAB function *igrf11syn* implements IGRF model. It expects four inputs:

1. fyears = date in fractional years e.g., 2010
2. alt = altitude in (km)
3. nlat = latitude positive north(deg)
4. elong = longitude positive east(deg)

and generates output Earth's magnetic field B = [B_North; B_East; B_up] in (nano Teslas). A MATLAB script is written which generates a IGRF look-up in form of a

[2] `https://www.ngdc.noaa.gov/IAGA/vmod/igrf.html`
[3] https://en.wikipedia.org/wiki/Zenith
[4] `https://en.wikipedia.org/wiki/Spherical_coordinate_system`
[5] `https://en.wikipedia.org/wiki/Geographic_coordinate_system`

header file. By assuming orbit to be spherical altitude is fixed at 6978. This assumption is made to reduce the size of look-up table which will be stored in microcontroller's flash memory. This MATLAB script spans over a range of longitude and latitude and generates a header file. This scrpit for floating and fixed-point can be seen in Appendix-B.

2.3.2 Direction of the Sun

The direction of Sun can be modeled using ECI reference frame. Earth makes one complete rotation, around Sun, per year. The start of spring is also called the moment of Vernal equinox. During this period the Sun is in the positive x-direction. The time since the last vernal equinox can be termed as t_{ve}. If this t_{ve} is known, then Sun direction $\boldsymbol{S_I}$ can be calculated using below equation-2.26:

$$\boldsymbol{S_I} = \begin{bmatrix} \cos\frac{2\pi(t-t_{ve})}{T_{year}} \\ \cos\epsilon \sin\frac{2\pi(t-t_{ve})}{T_{year}} \\ \sin\epsilon \sin\frac{2\pi(t-t_{ve})}{T_{year}} \end{bmatrix} \tag{2.26}$$

Where T_{year} is the total time of a year in seconds, i.e., $(365 * 24 * 60 * 60) = 31536000$s. The angle between the Earth's equatorial plane and the orbital plane of the Earth around the Sun is ϵ. And t is the current time in seconds. The distance between the center of the Earth and the center of the satellite is negligible compared to, that of, Earth and Sun. Hence, we can assume that the direction of the Sun from the center of the satellite, is the same as seen from the center of Earth. When seen from the satellite the Earth can sometimes block the Sun, and this orbit period is called an eclipse period. We can calculate this eclipse period of the satellite using, satellite's position vector, the Sun direction vector and the half angular size of the Earth γ_{Earth} in the satellite's orbit.

$$\gamma_{Earth} = \arcsin\frac{R}{R+h} = \arcsin\frac{6378}{6378+h} \approx 66^\circ \; at\, h = \; 600\, km \tag{2.27}$$

Where, the radius of the Earth, R =6378 km, and $h = 600\, km$ altitude of the Delf-N3xt's orbit. The angle between the satellite's position vector, $\boldsymbol{r}_{satI}$, and the Sun direction vector, $\boldsymbol{S_I}$, can be calculated as:

$$\alpha\angle rS = \arccos\frac{\boldsymbol{r}_{satI}.\boldsymbol{S_I}}{\|\boldsymbol{r}_{satI}\|} \tag{2.28}$$

The satellite is said to be in the eclipse period when:

$$\pi - \alpha\angle rS < \gamma_{Earth} \tag{2.29}$$

This eclipse period can be visualized by Figure 2.7.

2.4 Attitude estimation using EKF

Attitude estimation is similar to estimating the rotational matrix that describe the orientation in SBF reference frame with respect to a known reference frame. Generally,

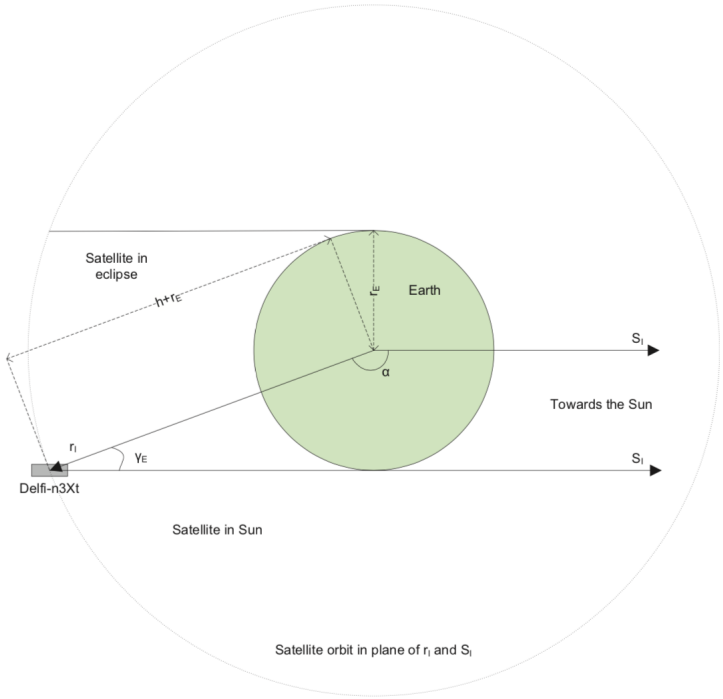


Figure 2.7: The geometry of the satellite during eclipse and non-eclipse period [1]

this known reference frame is ECI reference frame, or I frame. Both attitude and rotational rate are described with respect to I frame. Moreover, the reference vectors are also defined with respect to I frame. In general, by using various sensor measurements and their mathematical model, vector components within several reference frames can be collected. Later, these vector components are used in one of the estimation algorithms to determine the rotational matrix, in the form of a DCM, Euler angles or Quaternions [8], which were described in the section 2.2. Attitude estimation can be done using two types of algorithms. Firstly, deterministic algorithms where attitude can be determined using two or more vector observations from a single point in time. Secondly, recursive algorithms which use several sensors and combine these measurements with dynamic and kinematic models. In this thesis work, a recursive algorithm is used to estimate attitude of the satellite.

The Kalman filter is a statistical approach to discrete data filtering problem that offers optimal solutions for linear estimation problems. However, in practice, the physical processes are non-linear in nature. Hence, a modified version of the Kalman filter that linearizes the current covariance and the mean called as EKF is utilized. In the case of the Delfi-n3Xt satellite process to be estimated are non-linear and the ADCS uses EKF for attitude estimation. The EKF uses control inputs, sensor measurements, and the system's dynamical model to estimate the state of the system. All these measurements have noise and uncertainties inherently present, but EKF works in such a way that the estimated state has the lowest uncertainty. These uncertainties are modeled in an

error covariance matrix. The first step of EKF is to propagate the previous state to the current state, which is known as the prediction step or time propagation step. For the first prediction step, the previous state is modeled as an initial state. The time propagation step is described in the first subsection. The second step is measurement update. This step utilizes the previous estimates and combines them with the sensor measurements at the current time and updates the state. This measurement update step is described in the second subsection.

2.4.1 Time propagation

This subsection describes prediction step of EKF. In order to propagate state, from previous time to the current time, two different methods are presented. For the first propagation step, an initial estimate is required to act as previous estimate. Hence, we require an initial estimate for quaternion and rotational rate. Similarly, an initial process covariance matrix is also required.

For a given current estimates for quaternion and rotational rate $\hat{\boldsymbol{X}}_{k/k} = \left[\hat{\boldsymbol{q}}_{k/k}^T, \hat{\boldsymbol{\omega}}_{k/k}^T\right]$, the state can be propagated as:

$$\hat{\boldsymbol{\phi}}_{k/k} = \exp\left(\frac{1}{2}\hat{\boldsymbol{\Omega}}_{k/k}\Delta t\right) \tag{2.30}$$

$$\hat{\boldsymbol{q}}_{k+1/k} = \hat{\boldsymbol{\phi}}_{k/k}\hat{\boldsymbol{q}}_{k/k} \tag{2.31}$$

$$\hat{\boldsymbol{\omega}}_{k+1/k} = \hat{\boldsymbol{\omega}}_{k/k} + \hat{\dot{\boldsymbol{\omega}}}_{k/k}\Delta t \tag{2.32}$$

Where, $\hat{\boldsymbol{\Omega}}$ can be defined as:

$$\hat{\boldsymbol{\Omega}} = \begin{bmatrix} 0 & \omega_3 & -\omega_2 & \omega_1 \\ -\omega_3 & 0 & \omega_1 & \omega_2 \\ \omega_2 & -\omega_1 & 0 & \omega_3 \\ -\omega_1 & -\omega_2 & -\omega_3 & 0 \end{bmatrix} \tag{2.33}$$

using the rotational dynamics equation that was not derived in this document but directly stating from [1]:

$$\dot{\boldsymbol{\omega}} = \boldsymbol{J}^{-1}\left(\boldsymbol{T}_c + \boldsymbol{T}_d - \boldsymbol{\Omega}(\boldsymbol{J}\boldsymbol{\omega} + \boldsymbol{h})\right) \tag{2.34}$$

Where $\boldsymbol{T}_c$ is control torque, $\boldsymbol{T}_d$ is disturbance torques, $\boldsymbol{J}$ is the inertia matrix of the Delfi-n3Xt including the non rotating reaction wheels, $\boldsymbol{h}$ is angular momentum of the reaction wheels, and the gyroscopic effect that is cased by rotation of reaction wheels can be seen by $\boldsymbol{\Omega}\ \boldsymbol{h}$. $\hat{\dot{\boldsymbol{\omega}}}_{k/k}$ is the estimate of derivative of rotational rate, using above equation, can be calculated as:

$$\hat{\dot{\boldsymbol{\omega}}}_{k/k} = \boldsymbol{J}_{kf}^{-1}\left((\boldsymbol{m}_k \times \boldsymbol{B}_{mk}) - \hat{\boldsymbol{\Omega}}\left(\boldsymbol{J}_{kf}\hat{\boldsymbol{\omega}}_{k/k} + \hat{\boldsymbol{h}}_k\right)\right) \tag{2.35}$$

Where $\boldsymbol{J}_{kf}$ is the inertia matrix on-board on the Delfi-n3Xt, and this will not match with true inertia matrix. $\boldsymbol{m}_k$ is the magnetic dipole applied on magnetorquers during previous control step, $\boldsymbol{B}_{mk}$ is the magnetic field vector measured during the previous control step. $\hat{\boldsymbol{h}}$ is angular momentum of the reaction wheels. The other disturbance torques

are omitted from this equation for simplicity. Two main methods will be considered to propagate estimated error covariance matrix. Firstly an update method using Additive Extended Kalman Filter (AEKF) is presented. Secondly, Pseudo Linear Kalman Filter (PLKF) method is described.

Following matrix can be defined for AEKF method:

$$\boldsymbol{F}(\hat{\boldsymbol{\omega}}_{k/k}) = \begin{bmatrix} 0 & \omega_3 s_1 & \omega_2 s_1 \\ \omega_3 s_2 & 0 & \omega_1 s_2 \\ \omega_2 s_3 & \omega_1 s_3 & 0 \end{bmatrix} \tag{2.36}$$

Where, $\omega_i = \hat{\omega}_{k/k_i}$ for i= 1,2,3 and

$$\boldsymbol{s} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix} = \begin{bmatrix} \frac{J_{kf_{22}} - J_{kf_{33}}}{J_{kf_{11}}} \\ \frac{J_{kf_{33}} - J_{kf_{11}}}{J_{kf_{22}}} \\ \frac{J_{kf_{11}} - J_{kf_{22}}}{J_{kf_{33}}} \end{bmatrix} \tag{2.37}$$

Below matrix will is also required:

$$\hat{\boldsymbol{\Xi}}_{k/k} = \boldsymbol{\Xi}(\hat{\boldsymbol{q}}_{k/k}) = \begin{bmatrix} q_4 & -q_3 & q_2 \\ q_3 & q_4 & -q_1 \\ -q_2 & q_1 & q_4 \\ -q_1 & -q_2 & -q_3 \end{bmatrix} \tag{2.38}$$

where, q_i=q_{k/k_i} foe i = 1,2, 3,4. Using above matrices the below matrix can be defined:

$$\boldsymbol{\Psi}_k = \exp\left(\begin{bmatrix} \frac{\hat{\boldsymbol{\Omega}}_{k/k}}{2} & \frac{\hat{\boldsymbol{\Xi}}_{k/k}}{2} \\ \boldsymbol{0} & \boldsymbol{F}(\hat{\boldsymbol{\omega}}_{k/k}) \end{bmatrix} \Delta t \right) \tag{2.39}$$

Process noise matrix is as defined below:

$$\boldsymbol{P}_k^w = \begin{bmatrix} \sigma_q^2 I_4 & \boldsymbol{0} \\ \boldsymbol{0} & \sigma_\omega^2 \Delta t I_3 \end{bmatrix} \tag{2.40}$$

Here, σ_q and σ_ω are tuning parameters to improve the performance of EKF. Specifically, σ_q stands for standard deviation of process noise required to propagate the quaternion, and σ_q stands for standard deviation of process noise required to propagate the rotational rate. Thus, covariance matrix can be updated as:

$$\boldsymbol{P}_{k+1/k} = \boldsymbol{\Psi}_k P_{k/k} \boldsymbol{\Psi}_k^T + \boldsymbol{P}_k^w \tag{2.41}$$

But, a different $\boldsymbol{\Psi}$ matrix will be used for the PLKF method:

$$\boldsymbol{\Psi}_k = \exp\left(\begin{bmatrix} \boldsymbol{0} & \frac{\hat{\boldsymbol{\Xi}}_{k/k}}{2} \\ \boldsymbol{0} & \boldsymbol{F}_1(\boldsymbol{\omega})_{m/k} \end{bmatrix} \Delta t \right) \tag{2.42}$$

It is important to note that PLKF does not require linearization. It relies on the identity given below:

$$(\boldsymbol{J}\boldsymbol{\omega}) \times \boldsymbol{\omega} \equiv \boldsymbol{F}_1(\boldsymbol{\omega})\boldsymbol{\omega} \tag{2.43}$$

Where $\boldsymbol{F}_1(\boldsymbol{\omega})$ is given by:

$$\boldsymbol{F}_1(\boldsymbol{\omega}) = \begin{bmatrix} 0 & \sigma_1\omega_3 & 0 \\ 0 & 0 & \sigma_2\omega_1 \\ \sigma_3\omega_2 & 0 & 0 \end{bmatrix} \tag{2.44}$$

Using this matrix, a new $\boldsymbol{\Psi}$ matrix can be obtained, and the covariance matrix can be propagated using Equation 2.41.

2.4.2 Measurement update

This section will present measurement update step of EKF. Two different methods for measurement update are presented in [7]. It is important to note that the Delfi-n3Xt uses two sensors. But, the data from both the sensors is not available all the time. The two sensors used are magnetometer and Sun-sensor. Here, Sun-sensor data is not available when satellite is in eclipse period. It is also important to note that both the sensors still will use the same measurement update algorithm. In contrast, when there is no data available for Sun-sensor update, measurement update using Sun-sensor is skipped for that step.

The Quaternion Kalman Filter (QKF) method as described in [7] is presented first.

$$\boldsymbol{s}_{k+1} = \frac{1}{2}\left(\boldsymbol{b}_{k+1} + \boldsymbol{r}_{k+1}\right) \tag{2.45}$$

$$\boldsymbol{d}_{k+1} = \frac{1}{2}\left(\boldsymbol{b}_{k+1} - \boldsymbol{r}_{k+1}\right) \tag{2.46}$$

Where, $\boldsymbol{r}_{k+1}$ stands for reference vector with respect to inertial reference frame, and $\boldsymbol{b}_{k+1}$ stands for normalized measurement vector with respect to body reference frame. Using this observation matrix can be constructed as:

$$\boldsymbol{H}_{k+1} = \begin{bmatrix} -\left[\boldsymbol{s}_{k+1}\times\right] & \boldsymbol{d}_{k+1} \\ \boldsymbol{d}_{k+1}^T & \boldsymbol{0} \end{bmatrix} \tag{2.47}$$

$$\bar{\boldsymbol{H}}_{k+1} = \begin{bmatrix} \boldsymbol{H}_{k+1} & \boldsymbol{O}_{4\times 3} \end{bmatrix} \tag{2.48}$$

The sensor noise matrix can be defines as:

$$\boldsymbol{P}_{k+1}^v = \sigma_b^2 I_4 \tag{2.49}$$

Where, σ_b stands for standard deviation of sensor noise, as used by the filter. This value can be tuned to improve performance of the filter. Furthermore, state and covariance matrix can be updated using below equations [1]:

$$\boldsymbol{S}_{k+1/k} = \bar{\boldsymbol{H}}_{k+1}\boldsymbol{P}_{k+1/k}\bar{\boldsymbol{H}}_{k+1}^T + \boldsymbol{P}_{k+1}^v \tag{2.50}$$

$$\boldsymbol{K}_{k+1} = \boldsymbol{P}_{k+1/k}\bar{\boldsymbol{H}}_{k+1}^T\boldsymbol{S}_{k+1/k}^{-1} \tag{2.51}$$

$$\hat{\boldsymbol{X}}_{k+1/K+1} = \left(I_7 - \boldsymbol{K}_{k+1}\bar{\boldsymbol{H}}_{k+1}\right)\hat{\boldsymbol{X}}_{k+1/K} \tag{2.52}$$

$$\hat{\boldsymbol{q}}_{k+1/k+1} = \frac{\hat{\boldsymbol{q}}_{k+1/k+1}}{\|\hat{\boldsymbol{q}}_{k+1/k+1}\|} \tag{2.53}$$

$$\boldsymbol{P}_{k+1/k+1} = \left(I_7 - \boldsymbol{K}_{k+1}\bar{\boldsymbol{H}}_{k+1}\right)\boldsymbol{P}_{k+1/k}\left(I_7 - \boldsymbol{K}_{k+1}\bar{\boldsymbol{H}}_{k+1}\right)^T + \boldsymbol{K}_{k+1}\boldsymbol{P}^v_{k+1}\boldsymbol{K}^T_{k+1} \tag{2.54}$$

The second method proposed in [7] is Reduced Quaternion Kalman Filter (RQKF), is an adaptation of the measurement update equations presented above.

The Observation matrix H from Equation 2.47 can be rewritten as:

$$\boldsymbol{H} = \begin{bmatrix} 0 & s_3 & -s_1 & d_1 \\ -s_3 & 0 & s_2 & d_2 \\ s_1 & -s_2 & 0 & d_3 \\ -d_1 & -d_2 & -d_3 & 0 \end{bmatrix} \tag{2.55}$$

If we carefully look at the H matrix written above it has a rank 2. row1 and 2 are linearly independent. While, row3 = -column3, and row4=-column4. Hence, it is possible to reduce this matrix in order to reduce computational burden in the EKF. In oder to pick the 'richest' pairs of row. We can see that there are six 2 x 4 sub-matrices of H. Let us denote this with H_{ij}, where ij={ 43,42,41,32,31,21 }. Then the trace of $H^T_{ij}\ H_{ij}$ can be calculated for each H_{ij} matrix. The pair yielding the maximum trace can then be selected as the reduced measurement matrix. For instance in case of H_{12}, the reduced matrix is:

$$\bar{\boldsymbol{H}} = \begin{bmatrix} \boldsymbol{H}_{12} & \boldsymbol{0} \end{bmatrix} = \begin{bmatrix} 0 & s_3 & -s_1 & d_1 & 0 & 0 & 0 \\ -s_3 & 0 & s_2 & d_2 & 0 & 0 & 0 \end{bmatrix} \tag{2.56}$$

This reduced matrix can then be used in measurment update step, where the sensor noise matrix can now be defined as:

$$\boldsymbol{P}^v_{k+1} = \sigma_b^2 I_2 \tag{2.57}$$

For the filter to have previous estimates when the filter is just started, there is a need for initial estimates. Hence the table- lists the initial values that are used in EKF of Delf-N3xt.

Table 2.1: Initial estimate values for EKF

Parameter	Value
$\hat{\boldsymbol{q}}_0\,[-]$	$[-0.415120208428812, -0.679093418021986, -0.341331718645127, 0.5]^T$
$\hat{\boldsymbol{\omega}}_0\left[deg\ s^{-1}\right]$	$[0,0,0]^T$
$\hat{\boldsymbol{P}}_0\,[-]$	$diag\left(\left[1e^{-2}, 1e^{-2}, 1e^{-2}, 1e^{-2}, 1e^{-2}, 1e^{-2}, 1e^{-2}\right]\right)$

2.5 Control algorithms

This section describes two different control algorithms that are used in the Delfi-n3Xt. First is Bdot controller which is a basic controller that requires magnetometer and magnetorquer to stabilize the satellite. Second is Quaternion feedback regulator that may

use more than one sensor data to estimate the current attitude and rotational rate. And may use more than one actuator to control the satellite. Since, Bdot is a less intensive algorithm and requires only one sensor and actuator it is considered to be more robust and reliable.

2.5.1 Bdot controller

As the satellite is ejected from the launch container for the Delfi-n3Xt it was expected to have atleast 10 degrees per second initial rotational rate of satellite around each axis, and this had to brought to below one degree per second as explained in introduction chapter. If this is not achieved the satellite mission can fail. Hence, detumbling has to be achieved by a failure free and robust algorithm.

In principle Bdot algorithm is a simple algorithm. Magnetometer on-board performs magnetic field measurements, and this magnetic field vector is measured in SBF reference frame. Using a minimal computation Bdot algorithm generates control torque that is used by magnetorquers to control the satellite. The control law equation which is proposed in [12], is given by:

$$\boldsymbol{m}_{\dot{B}} = -k_{\dot{B}}\dot{\boldsymbol{B}}_B \tag{2.58}$$

Where $\boldsymbol{m}_{\dot{B}}$ refers to the magnetic dipole vector that has to be generated by magnetorquers in order to achieve detumbling. $k_{\dot{B}}$ is the positive controller gain which is dependent on expected rotational rate of satellite. And, $\dot{\boldsymbol{B}}_B$ is the time derivative of magnetic field vector in SBF reference frame. The torque that is acted upon satellite due to the magnetic dipole vector produced by magnetorquers to contract tumbling of satellite is given by:

$$\boldsymbol{T}_{\dot{B}} = \boldsymbol{m}_{\dot{B}} \times \boldsymbol{B}_B \tag{2.59}$$

Where, $\boldsymbol{B}_B$ is the magnetic field vector in SBF reference frame. $\dot{\boldsymbol{B}}_B$ the time derivative of magnetic field vector in SBF reference frame is given by:

$$\dot{\boldsymbol{B}}_B = \dot{\boldsymbol{B}}_I + \boldsymbol{\omega}^{I/B} \times \boldsymbol{B}_B = \dot{\boldsymbol{B}}_I - \boldsymbol{\omega}^{B/I} \times \boldsymbol{B}_B \tag{2.60}$$

Where, $\boldsymbol{\omega}^{B/I}$ is the rotational rate of the SBF reference frame (B frame) with respect to ECI reference frame (I frame). Similarly, $\boldsymbol{\omega}^{I/B}$ is rotational rate of I frame with respect to B frame. $\boldsymbol{\omega}^{B/I} \equiv \boldsymbol{\omega}$ can be referred to as body rotational rate in short.

But, the Delfi-n3Xt requires a discretized algorithm and the following equations are implemented on board. The magnetic field in B frame is measured by magnetometer during each control loop i, where $\boldsymbol{B}_{\boldsymbol{Bm}i} \equiv \boldsymbol{B}_{mi}$. The time derivative of magnetic field measurements can be calculated using the two consecutive measurements (control steps) given by:

$$\dot{\boldsymbol{B}}_i = \frac{\boldsymbol{B}_{mi} - \boldsymbol{B}_{mi-1}}{\Delta t} = \frac{\Delta \boldsymbol{B}_m}{\Delta t} \tag{2.61}$$

Where, Δt is the time step, which is of 2 seconds. With the calculated value of $\dot{\boldsymbol{B}}_i$ and using Equation 2.58 the desired magnetic dipole is given by:

$$\boldsymbol{m}_{\dot{B}i} = -k_{\dot{B}}\dot{\boldsymbol{B}}_i \tag{2.62}$$

In order to deliver the control torque given by:

$$\boldsymbol{T}_{\dot{B}i} = \boldsymbol{m}_{\dot{B}i} \times \boldsymbol{B}_{Bi} \tag{2.63}$$

The $k_{\dot{B}}$ is choosen in such a way that the magnetorquers are fully utilized and for the Delfi-n3Xt this is calculated as $k_{\dot{B}} = 50000\, A\, m^2\, s\, T^{-1}$ [1]. To understand more in detail how the $\boldsymbol{T}_{\dot{B}i}$ influences the rotation of satellite readers are referred to [1].

2.5.2 Quaternion feedback regulator

In FSP mode we require much more advanced controller than Bdot to calculate control torque that will rotate the satellite to the desired attitude much more accurately. The Delfi-n3Xt uses control law equation proposed in [13] for calculating torque in Quaternion feedback regulator. The following is the control law equation to calculate the control torque:

$$\boldsymbol{T}_{control} = -d\boldsymbol{J}\boldsymbol{\omega}_e - k\boldsymbol{J}\boldsymbol{q}_e + \hat{\boldsymbol{\Omega}}\boldsymbol{I}\hat{\boldsymbol{\omega}} \tag{2.64}$$

Where k is proportional gain, d is differential gain, $\boldsymbol{J}$ is the moment of inertia matrix of the satellite, error between desired and estimated inertial rotational rate is $\boldsymbol{\omega}_e$, error between desired and estimated quaternion is $\boldsymbol{q}_e$, estimated rotational rate of satellite it $\boldsymbol{\omega}$, and skew symmetric matrix constructed using estimated rotational rate is $\hat{\boldsymbol{\Omega}}$. Error between desired and estimated rotational rate $\boldsymbol{\omega}_e$, can be calculated using desired rotational rate (0° on each axis) $\boldsymbol{\omega}_d$ and estimated rotational rate $\hat{\boldsymbol{\omega}}$, and the desired inertial attitude quaternion $\boldsymbol{q}_d$ and the estimated inertial attitude quaternion $\hat{\boldsymbol{q}}$. The error in rotational rate is given by:

$$\boldsymbol{\omega}_e = \hat{\boldsymbol{\omega}} - \boldsymbol{\omega}_d\boldsymbol{q}_e = \begin{bmatrix} q_{d4} & q_{d3} & -q_{d2} & -q_{d1} \\ -q_{d3} & q_{d4} & q_{d1} & -q_{d2} \\ q_{d2} & -q_{d1} & q_{d4} & -q_{d3} \end{bmatrix} \hat{\boldsymbol{q}} \tag{2.65}$$

In principle $\hat{\boldsymbol{\Omega}}\boldsymbol{I}\hat{\boldsymbol{\omega}}$ is added to counteract the gyroscopic coupling effect, that is created by satellite during rotation. In case of the Delfi-n3Xt satellite, which will not be rotating during normal operation, this term adds computational complexity with no additional control accuracy [1]. Hence, this term is neglected and control torque equation can be rewritten as:

$$\boldsymbol{T}_{control} = -d\boldsymbol{J}\boldsymbol{w}_e - k\boldsymbol{J}\boldsymbol{q}_e \tag{2.66}$$

The scalar gain parameters k and d determine the settling and damping time of the control algorithm. The control torque $\boldsymbol{T}_{control}$ has to be delivered using both or either of actuators, magnetorquer and reaction wheels. For the Delfi-n3Xt satellite, this load division of $\boldsymbol{T}_{control}$ is done by following scheme:

$$\boldsymbol{T}_{control} = \boldsymbol{T}_{rw} + \boldsymbol{T}_{mtq} = -\dot{\boldsymbol{h}}_c + \boldsymbol{m}_c \times \boldsymbol{B} \tag{2.67}$$

The magnetic dipole that needs to delivered by magnetorquers, $\boldsymbol{m}_c$, is given by:

$$\boldsymbol{m}_c = -\frac{\boldsymbol{T}_{control} \times \boldsymbol{B}}{\|\boldsymbol{B}\|^2}. \tag{2.68}$$

The acceleration which has to be delivered by reaction wheels is given by:

$$\dot{\boldsymbol{h}}_c = (\boldsymbol{m}_c \times \boldsymbol{B}) - \boldsymbol{T}_{control} \tag{2.69}$$

This load division scheme is used in FSP mode.

Controlling of the satellite using reaction wheels requires a technique of angular momentum unloading (i.e., by dumping of stored angular momentum). In order to deliver the required control torque magnetorquers and reaction wheels are accelerated/decelerated. Because of this reaction wheel can store an angular momentum. But, reaction wheels can only store a certain maximum amount of angular momentum. Thus, the reaction wheels must be unloaded on a regular basis. For more detailed explanation readers are referred to [1].

2.6 Power budget for ADCS in the Delfi-n3Xt

This section presents the power budget of the Delfi-n3Xt. It also presents power budget for ADCS in the Delfi-n3Xt. It is important to note that the Delfi-n3Xt uses a traditional ATxmega128A1 microcontroller running at 32 Mhz. At the time this satellite was launched this was the low power and high performance microcontroller from Atmel [1]. The Table 2.2 shows the power budget for the Delfi-n3Xt. Using Table 2.2 it can be

Subsystem	Mode	Power requirement (mW)
$T^3\mu PS$ (micro Propulsion Subsystem)	On (Measure Only)	45
SDM (Solar cell Degradation Measurement)	on	84
ADCS	3-Axis control(ADCS1)	1665
PTRX (Primary Transceiver)	transceive	1745
ITRX (ISIS Transceiver)	receive	256
STX (S-band Transmitter)	store only	75
OBC (On Board Computer)	on	234
EPS (Electric Power Subsystem)	MPPT	1048
DAB (Digital Audio Broadcast)	Idle	84
Total	Nominal	5236

Table 2.2: Power budget for nominal satellite mode in the Delfi-n3Xt [4]

calculated that $\approx$ 32% of total nominal power was allocated to ADCS. The technical report [4] provides detailed power budget for ADCS, as shown in Table 2.3. It can be calculated, that $\approx$ 32 % of power is allocated to ADCS software, with respect to total ADCS power budget. This is $\approx$ 10 % with respect to total nominal power budget for the Delfi-n3Xt.

2.7 Summary

This chapter describes the ADCS and different algorithms that are used within the Delfi-n3Xt. It also enlists that $\approx$ 10 % of the Delfi-n3Xt, total nominal power budget, is allocated to ADCS software.

Subsystem	Power requirement (mW)
Sun-sensor	206
Magnetorquer system	347
Reaction Wheel system	519
Main Board ADCS 1	535
Main Board ADCS 2	0
DSSB	58
Total	1665

Table 2.3: ADCS power budget [4]

Digital Signal Processing Alternatives

3

To efficiently implement ADCS it is essential to study DSP alternatives. Two well-known categories can be used to store and manipulate numeric representations of data: Fixed-Point (FxP) and floating-point. The floating-point refers to data representation where the decimal point can float around a minimum of 32 bits. Whereas, in FxP decimal point is fixed at a position that determines the range and precision of representation. For example, based on word length, a FxP can represent numbers such as 123.45, 123.456, 123.4567, 123.15678, etc. Whereas, floating-point can float its decimal point and represent numbers such as 1.234567, 123456.7, 0.00001234567, 1234567000000000, etc. Hence, the floating-point can represent a much wide range of values as compared to the FxP.

Considering the precision requirement in pointing accuracy although it seems appealing to use floating-point arithmetic, it might not be the best possible solution considering the energy requirements. In such case, FxP arithmetic is promising alternative. But long development time, and effort required for scaling of variables, in order to, prevent overflow while maintaining the accuracy requirements makes it challenging [14]. Knowing the range and right precision required for an algorithm can be a tedious task. But, the FxP designer library from MathWorks [1] makes this exploration fast, and interactive. Need for rounding or saturation arithmetic, ranges of variables, a histogram to choose precision with the lowest error is made handy by this tool.

The most openly available FxP libraries like libfixmath [2] focus on only one FxP representation. Such representation attributes to a fixed precision and range. As the requirement for each variable can have multiple ranges and precision, in general not all the designs can be benefited from such fixed representation. Hence, there is a need to design a FxP library which addresses this problem. Such a library can not only benefit this thesis work but also, in general, any future FxP DSP. In this thesis work, a FxP library is developed.

This chapter will describe three different DSP alternatives. Two of which are in floating-point and third in FxP representation. It also explains the extension of an openly available FxP numeric library [3]. This library implements a basic 32-bit and 64-bit precision. It uses 64bit arithmetic which requires 128-bit (long long) compiler. But, in general microcontroller compilers do not support this. Hence, this library was extended to use many different FxP precision possibilities and requiring a 64-bit compiler. First, this chapter will begin with floating-point design alternatives. Floating-point design uses IEEE standard for representation. Secondly, this chapter will follow with the FxP design. The FxP design use integer representation. However, this integer representation is not straightforward. Hence, the FxP section describes this in detail. These operations

[1] `https://nl.mathworks.com/help/fixedpoint/`

[2] `https://en.wikipedia.org/wiki/Libfixmath`

[3] `https://sourceforge.net/projects/fixedptc/`

include addition, multiplication, division, etc. Moreover, these are the operations used in conversion of the Delft-n3Xt library to FxP. Lastly, an overview of extension in FxP library to accommodate many different FxP representations is explained.

3.1 Floating-point arithmetic

In DSP floating-point, arithmetic is used in the computation of data, in the form of real numbers. These real numbers are represented using an approximation, in order to support a trade-off between range and precision. Floating-point numbers are typically represented in the form of scientific notation. It generally has a form of $F * r^E$, where F is a fraction, E is an exponent of a certain radix r.

In modern computers floating-point numbers are represented using IEEE 754 standard. floating-point numbers are represented in the form of scientific notation with radix 2, represented as $F * 2^E$. Similarly, F denotes fraction part and E denotes exponent of radix 2. Most importantly, as we use a fixed Word Length (WL), for example, 32 or 64 bits, floating-point numbers suffer from loss of precision. For example, there can be an infinite set of real numbers between 0 to 0.5. But, a n bit binary number can only represent a finite set of 2^n numbers. Hence, not all the real numbers can be represented. A nearest approximation is used which gives rise to a loss of accuracy [4]. Floating-point arithmetic using IEEE 754 standard, can be categorized in following four types [5]:

- **Single precision (SP)**: This is represented using *float* in C language (not guaranteed). It uses 4 bytes of data storage. Its fraction has a precision of 24 bits.
- **Double precision (DP)**: This is represented using *double* in C language (not guaranteed). It uses 8 bytes of data storage. Its fraction has a precision of 53 bits.
- **Double extended**: This is represented using *long double* in C language (not guaranteed). It uses 80 bits of data storage. Its fraction has a precision of 64 bits.
- **Quadruple precision**: This is represented using *__float128* in C language (not guaranteed). It uses 16 bytes of data storage. Its fraction has a precision of 113 bits.

This thesis uses only SP and DP arithmetic. Therefore only these two are explained in following subsections. It is important to note that it only gives an overview. This thesis uses the representation in IEEE 754 standard floating-point arithmetic, using C language. These are implemented using a standard Floating Point Unit (FPU), in ARM-M4 processor. In detailed understanding of floating-point arithmetic is not considered to be essential for this thesis. Readers are referred to [15] for in detailed explanation of each arithmetic operation.

[4] `http://www.ntu.edu.sg/home/ehchua/programming/java/datarepresentation.html`
[5] `https://en.wikipedia.org/wiki/Floating-point_arithmetic`

3.1.1 Single precision arithmetic

Consider a single precision floating-point representation as shown in Figure 3.1: with the 32-bit pattern as 1 1000 0001 000 0100 0000 0000 0000 0000, with:

- S = 1
- E = 1000 0001
- F = 000 0100 0000 0000 0000 0000

Sign bit represents the sign of the number 1 represents signed number, whereas 0 represents unsigned number.
In order to normalize, the actual fraction is added with an implicit leading 1. For this example, the actual fraction is $1.000\,0100\,0000\,0000\,0000\,0000 = 1 + 1 * 2^{-5} = 1.03125$
In order to normalize, the actual exponent ($1 \leqslant E \leqslant 254$) is biased with -127, to represent in rage (-127, $+128$). In this example $E - 127 = 129 - 127 = 2D$.
Hence, the number represented is $-1.03125 * 2^2 = -4.125D$.

But, normalization has a serious problem that is zero cannot be represented. Hence, let us consider denormalized form: For numbers with E=0, the denormalized form is used. Here, the actual exponent is always -126, and an implicit leading 0 is added instead of 1 for the fraction. Hence, the number zero can be represented using $E = 0$ and $F = 0$, as $0.0 * 2^{-126} = 0$ [4]. Using this form we can also represent very small numbers. For example, if $S = 1$, $E = 0$, and $F = 011\,0000\,0000\,0000\,0000\,0000$. The actual fraction can be seen as $0.011 = 1 * 2^{-2} + 1 * 2^{-3} = 0.375D$. Since, $S = 1$, it is a negative number. With $E = 0$, the actual exponent is -126. Here, the number can be seen as, $-0.375 * 2^{-126} = -4.4 * 10^{-39}$. Hence, this is a very small number close to 0.

Lastly, $E = 255$ can be used to represent special values such as $\pm\infty$ and NaN (not a number), which are out of scope for this document.

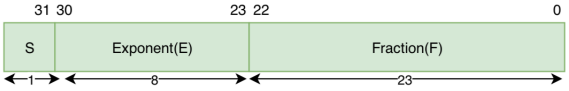


Figure 3.1: single precision floating-point representation

3.1.2 Double precision arithmetic

Similarly, double precision arithmetic representation is shown in Figure 3.2.In contrary to single precesion here the WL is 64-bit where S = 1-bit, E = 11-bit and F = 52-bit. Here value of number N can be calculated as:

Figure 3.2: double precision floating-point representation

- Normalized form: For $1 \leqslant E \leqslant 2046$, $N = (-1)^S * 1.F * 2^{(E-1023)}$.
- Denormalized form: For $E = 0$, $N = (-1)^S * 0.F * 2^{(E-1022)}$.
- Special valuse: $E = 2047$, N used to represent special values such as $\pm\infty$ and NaN (not a number), which are out of scope for this document.

floating-point arithmetic is usually accelerated using a dedicated co-processor called as FPU. But, not all microprocessors have a dedicated FPU.

3.2 FxP arithmetic

As explained earlier not all microprocessors include FPU. Hence, DSP using floating-point arithmetic becomes difficult. There are compiler based alternatives which are computationally expensive. Furthermore, DSP co-processor are widely used to accelerate DSP computations. These processors however only support integer/FxP arithmetic. Moreover, old processors do not include FPU. When a processor includes FPU it increases area and power to add this capability. Hence, under many circumstances FxP DSP computation are preferred over floating-point computation. FxP computation can be used to provide improved performance or accuracy for an application. For example consider ARM-M3 which does not feature an FPU or ARM-M4F which does. In terms of area, ARM-M3 uses $12mm^2$, where as ARM-M4F uses $0.17mm^2$. In terms of power, ARM-M3 is $1\mu W/Mhz$ more efficient than ARM-M4F. Hence, ARM-M3 could be a better platform, if power and area are both considered more important than performance. However, ARM-M3 could outperform over ARM-M4F (as $CoreMark/MHz$ for, ARM-M4F is 0.08 better than ARM-M3) [6].

3.2.1 FxP notation

In order to represent FxP data the following generalized format, employed by [14] is used:

$$< word\_length, integerword\_length > \tag{3.1}$$

FxP notation considers the presence of a hypothetical binary point. Left of this binary point is referred to as Integer Word Length (IWL) which determines the maximum and minimum range of integer number that can be represented. The right of binary point is referred as Fraction Word Length (FWL) which determines the minimum quantization step possible for the representation. The total number of bits that are used in this representation is referred to as WL.

Hence, range for a FxP notation can be determined based on IWL:

$$-2^{\mathrm{IWL}} \leqslant R \leqslant 2^{\mathrm{IWL}} \tag{3.2}$$

and Quantisation steps (QS) can be determined by:

$$Q = 2^{-\mathrm{FWL}-1} = 2^{-(\mathrm{WL}-\mathrm{IWL}-1)} \tag{3.3}$$

[6] https://images.anandtech.com/doci/8400/Screen%20Shot%202014-08-18%20at%206.03.23%20PM.png

For example, a 32bit WL having the binary point in middle i.e., IWL=16 and FWL=16 can be represented as $< 32, 16 >$ in the FxP notation. It has a range of $(-2^{15}, 2^{15})$ and QS of 2^{-15}. An initial study was done by varying FWL to see the errors of variable in EKF propagation step. Although a WL of 32 or 64bits can be used in microcontrollers choosing a FWL for this implementation with lowest error can be benefited by such a study. Also, custom hardware development in future using for example, an Field-Programmable Gate Array (FPGA) can use any arbitrary WL and may have a benefit. But, doing such a study for complete ADCS algorithm is tedious and time consuming. In Delfi-Nxt ADCS algorithm, maximum number of variables use a range of IWL=4. Hence, using a 32bit WL use of FxP representation $< 32, 28 >$ is proposed. But, to prevent an overflow for other variables, the same representation could not be used. Hence, a second precision of $< 64, 28 >$ is proposed. However, an arbitrary precision can also be used to prevent overflow.

3.2.2 Basic arithmetic used in FxP numeric library

The Delfi-n3Xt ADCS algorithm uses intensive matrix DP arithmetic operations. In order to use FxP, the matrix library from the Delfi-n3Xt is rewritten in FxP. All the matrix operation such as matrix multiplication, addition, division, square root, exponential, transpose, inverse, etc were rewritten to use FxP arithmetic. There are certain arithmetic rules to be followed in order to produce valid results in FxP. This section introduces to such rules and demonstrates basic FxP arithmetic to produce meaningful results. Moreover, the arithmetic resources as explained below are used to rewrite Delfi-n3Xt matrix library, using FxP.

3.2.2.1 Assignment

In order to perform assignment from one FxP variable to other it is essential that the decimal point remains in right place. In order to demonstrate let us consider two variables x and y. If x has an IWL of 1 and y has IWL of 2 they cannot be directly assigned. Right or left shift has to be done before assigning as seen in Figure 3.3. For this example, $y = x$ should be performed as $y = x >> 1$, and $x = y$ should be performed as $x = y << 1$.

3.2.2.2 Addition and subtraction

In order to perform addition and subtraction using FxP, both the operands must have decimal point at the same place. Moreover, decimal place of result must also be considered before performing the operation. In order to demonstrate let us consider two variables x and y. If x has an IWL of 1 and y has IWL of 2 they cannot be directly added. In case, the IWL of result is 2, then x must right shifted once i.e., $x + y = (x >> 1) + y$ Figure 3.4. Similarly, if the IWL of result is 3, then $x + y = (x >> 2) + (y >> 1)$.

3.2.2.3 Multiplication

In order to perform multiplication using FxP, unlike other operations it is not necessary that operands must have decimal point at the same place. Moreover, the decimal point

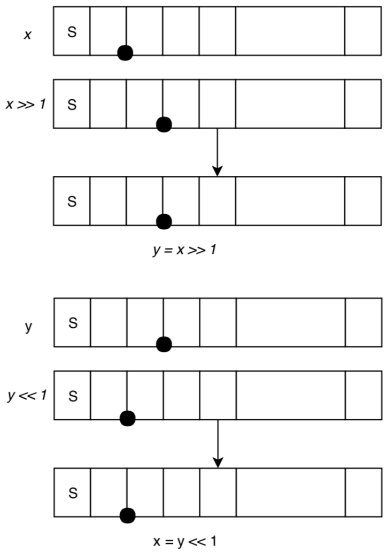


Figure 3.3: Assignment operation

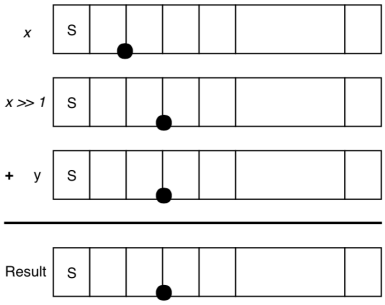


Figure 3.4: Addition operation

of the result is determined based on the decimal point of both the operands. In other words, IWL of the result is the sum of IWL's from both the operands plus one, as shown in Figure 3.5 the result has two sign bits. Multiplication of operands with WL of w's produce a product of $2w$. In general ANSI C compilation, the lower half of product is considered. But, in DSP algorithms most of the operands are aligned towards the left in order to provide maximum precision. Hence, in FxP multiplication, if we want to truncate our result to w bits instead of using $2w$ product, the upper part of w must be considered. But, a simple rounding technique could help increase the accuracy of such truncation. This thesis implements multiplication with a simple rounding technique. This rounding technique checks the first rightmost bit that is discarded in truncation, and if this bit is set then the product is simply incremented by one, or the product is unchanged. C-code for this rounding scheme is shown in Appendix A.5. However, it is important to note that this is a simple rounding technique which do not specifically round

negative numbers. In other words, this rounding technique could result in improper rounding of negative numbers. For future implementations if the FxP accuracy needs to be improved this rounding technique could be made more precise (which also could result in extra overhead).

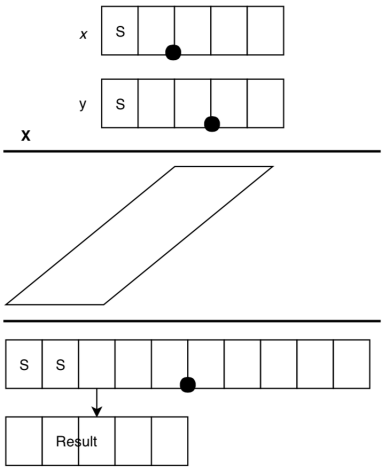


Figure 3.5: Multiplication operation

3.2.2.4 Division

In order to perform division using FxP, similar to multiplication it is not necessary that operands must have decimal point at the same place. Moreover, IWL and FWL of result is determined by IWL of first operand, and FWL of both the operands. In other words, result has $\mathrm{IWL}_r = \mathrm{IWL}_1 + \mathrm{FWL}_2$ and $\mathrm{FWL}_r = \mathrm{FWL}_1 - \mathrm{FWL}_2$, where $\mathrm{FWL}_1 > \mathrm{FWL}_2$ (Here subscript $_1$ and $_2$ indicate operand number, and $_r$ indicates result). Figure 3.6 demonstrates this example.

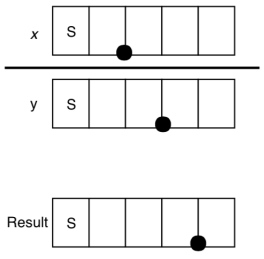


Figure 3.6: Division operation

3.2.2.5 Trigonometric functions

ARM libraries for ARM-M3 processor implements a look-up table based approach for *sin* and *cos* functions. These were readily utilized for sin, cos and *tan* functions. The Delfi-n3Xt ADCS uses inverse *tan* and *cos* function, but these are not supported by ARM libraries. COordinate Rotation DIgital Computer (CORDIC) algorithm uses only addition, subtraction, bit-shift and look-up table to implement these inverse trigonometric functions. An implementation for this algorithm is openly available at [7]. Hence, CORDIC algorithm with its simple and efficient calculations was utilized to perform inverse trigonometric functions. For detailed explanation of CORDIC algorithm readers are referred to [8].

3.3 Extensions for using different FxP precisions

A similar technique as explained above is used to extend different precision of arithmetic resources. It is important to note that it can be a tedious task to determine the number and type of shift for meaningful FxP arithmetic. The FxP library implemented in this thesis uses macros for simplicity. Hence, by using these pre-written macros FxP library can flawlessly perform arithmetic for arbitrary precision. For example, consider a macro which converts a floating-point number into FxP, as shown in Appendix A.1, simply by changing the FIXEDPT_FBITS variable with required FWL a floating-point number can be converted to FxP with required precision.

In order to switch from one FWL to another, a simple macro can perform the required shifts, and the user do not have to take care of type and number of shift to be performed. The Appendix A.2 shows a macro that performs conversion from FWL 28 to FWL 22. Utilizing the previously described $fixedpt\_rconst$ macro various variables can be defined, for example π can be defined as in Appendix A.3.

Multiplication can be achieved by macro shown in Appendix A.4. Here, by varying FIXEDPT_FBITS multiplication of a new FWL can be done. Note that this macro assumes both of its operands to have decimal point at the same position, if not FWL conversion macro is to be used in order to make them compatible.

Similarly, division can be achieved by macro shown in Appendix A.6. And, FIXEDPT_FBITS can be varied in order to perform division of a new FWL. Note that this macro also assumes both of its operands to have decimal point at the same position, if not FWL conversion macro is to be used in order to make them compatible.

The algorithm, as shown in Figure 3.7, is used to multiply two 64bits numbers by using 16 and 32bits arithmetic and produces a result of 128bits. This makes it possible to perform 64bits arithmetic and not require a 128bit compiler support. Here, operands a and b are split in 4 chunks of 16bits. For example, a can be split into, a_0, a_1, a_2, and a_4.

[7] `http://www.dcs.gla.ac.uk/~jhw/cordic/`

[8] `https://www.allaboutcircuits.com/technical-articles/an-introduction-to-the-cordic-algorithm/`

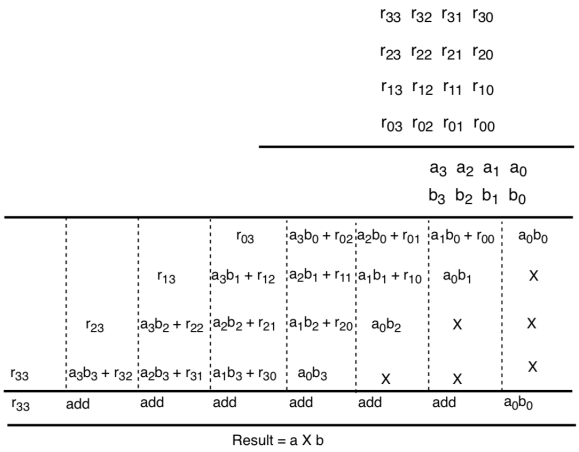


Figure 3.7: 64bits Multiplication using 16 and 32bits arithmetic

3.4 Summary

This chapter describes three different DSP alternatives. In particular DP, SP and FxP are explained. It is concluded that if compute power is considered more important, than accuracy, FxP computation can be considered as a better alternative. The extension of FxP library to support various FxP representations, using a 64 bit compiler support is explained. The FxP library implemented in this thesis, is used to port the Delfin3Xt matrix library from floating into FxP. An overview of these FxP operations are presented. In addition, extension of this FxP library in order to accommodate various FxP representation is explained.

Hardware in a Loop Simulation and Optimization

4

Chapter 1 presents an overview of research methodology used in this thesis. Figure 1.3 shows a block diagram that includes MATLAB and MSP-EXP432E401Y launchpad to complete the ADCS closed-loop implementation. For simplicity, this diagram is presented again in Figure 4.1. MATLAB environment marked in yellow implements the sensor, actuator, disturbance torques, attitude Kinematics and Dynamics. Where as, MSP-EXP432E401Y launchpad marked in red implements control and estimation. Chapter 2 ADCS algorithm describes the Delfi-n3Xt control and estimation in detail . But, in order to verify implementation control and estimation, the MATLAB models are also essential. If the MATLAB models marked in yellow, in Figure 4.1, are not used, then, the real sensor and actuators along with a virtual test-bench for simulating space environment is required. This thesis uses the MATLAB models that can simulate this effectively. Hence, by combining the effectiveness of MATLAB and ADCS implementation and MSP-EXP432E401Y launchpad, this thesis eliminates the need for sensor and actuator hardware.

In order to explain this in more detail, this chapter introduces to the technique that are used. The data exchange from/to MATLAB environment and MSP-EXP432E401Y launchpad board are required to perform hardware in loop simulation. The TCP/IP protocol is used to perform the data transmissions. MATLAB is modeled as a TCP/IP client and MSP-EXP432E401Y launchpad as a server. As explained in Chapter 1 this thesis will not implement all the modes that were designed for the Delfi-n3Xt. It only implements two of such modes which are presented in this chapter. Compiler optimization and DSP optimization techniques can be used to improve the performance of ADCS. This chapter explains such optimization techniques that are used in this thesis.

This chapter explains implementation and optimization techniques used in this thesis. Firstly, it begins by introducing to TCP/IP protocol. Secondly, describes the implementation of TCP/IP server/client software used in this thesis. Thirdly, it presents two modes of estimation and control, that are chosen for implementation. Thirdly, it presents advantages of using hardware in a loop simulation. Lastly, optimization techniques used to improve the performance of ADCS are explained.

4.1 Introduction to TCP/IP protocol

The TCP/IP protocol consists of several different protocols. Two such protocols are considered to be important. Hence, based on the importance of this two protocols, abbreviation for this protocol is derived as, TCP/IP. The Internet Protocol (IP) is the primary OSI model network layer (layer 3) protocol that provides addressing, datagram routing, and other functions in an internetwork. The Transmission Control Protocol (TCP) is the primary transport layer (layer 4) protocol and is responsible for connection

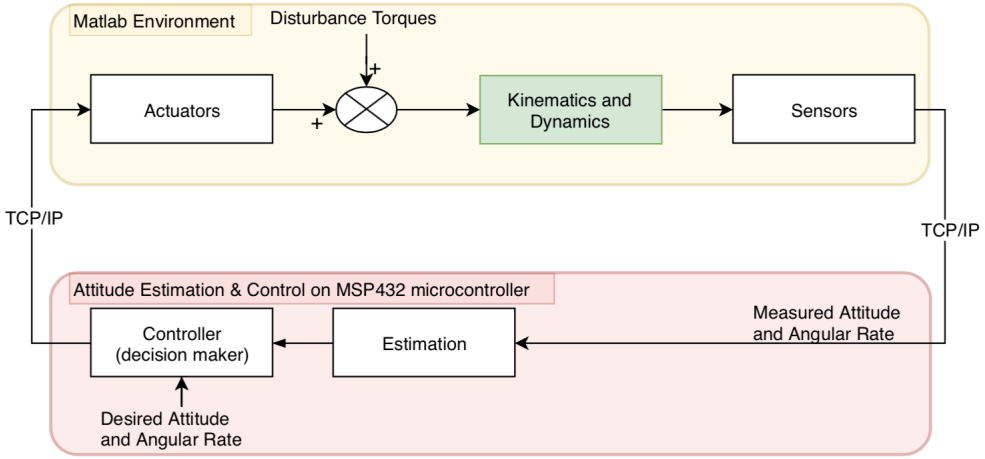


Figure 4.1: Simulation of ADCS with hardware in a loop

establishment and management, and reliable data transport between software processes on devices [16]. TCP/IP uses its own four-layered architecture (almost similar to the OSI Reference Model) for providing a framework for the various protocols. The TCP/IP protocol suite uses the notion of client/server based network communication. Clients normally initiate communications by sending requests, and servers respond to such requests, providing the client with the desired data or an informative reply such as error message. Although it is important to understand TCP/IP protocol, in dept theoretical understanding of layers and its functionality is not important for this thesis. Thus, for more detailed explanation readers are referred to an excellent guide [16].

4.2 Implementation of TCP/LWIP server/client software

This section describes implementation technique that are used for implementation of TCP/IP based server/client, in this thesis. In order to reduce the development time of a software, openly available software can be modified. One of the important reason for using openly available software is to reduce possibility of bugs. The openly available software are also maintained with frequent bug-fix, and are handy to utilize as a starting point. As stated above MATLAB is used as a client, and MSP-EXP432E401Y launchpad as a TCP/IP server. MATLAB TCP/IP client example explains this client implementation in detail, readers are referred to MATLAB documentation for more explanation [1]. Similar steps, as stated in this example are used in implementation of MATLAB TCP/IP client. Below is a list of task that are performed by this client:

- Creating a socket with the tcpip() system function.
- Setting address and port number of server.

[1] `https://nl.mathworks.com/help/instrument/tcp-ip-and-udp-interface.html`

- Connecting the socket to the address of the server using the fopen() system call.
- Sending and receive data by using fwrite/fread system call.
- Closing the socket at the end by using fclose() system call.

MSP-EXP432E401Y launchpad is used as a TCP/IP server and to implement estimation and control of ADCS. Resource explorer of TI provides a wide range of examples, one of which is tcpecho example [2]. This example software uses TI-RTOS kernel with a soft TCP/LWIP stack. It uses two tasks:

- **tcpecho**: This task creates a socket and accepts incoming connections. When a connection is established a tcpWorker task is dynamically created to send or receive data [17].
- **tcpWorker**: This task is responsible for sending or receiving data from/to client.

A function *main_sunpointing*() is implemented as a part of tcpWorker task which implements control and estimation of ADCS algorithm. This function implements the ADCS algorithm extended from the Delfi-n3Xt. Section 1.4 explains more details on the steps used for this extension. The ADCS used in this thesis is divided into two modes which are explained in next section.

4.3 Estimation and control modes of operation

This section describes two different estimation and control modes of operation that are considered in this thesis. As depicted in Figure 1.2 this thesis implements two modes marked in green. Firstly, detumble mode once the satellite is ejected from launch container detumbling mode must bring the satellite rotational rate to, less than 1 $^\circ$ /s. Once this tumble rate has been achieved satellite can switch from detumble mode to FSP mode. Secondly, FSP mode uses EKF algorithm using both sun sensor and magnetometer sensor data to point solar-panels towards sun. It is only allowed to have a maximum of 25 degree, sun-pointing error. Figure 4.2 shows blocks representing these two modes. Chapter-2 explains Bdot controller and Quaternion Feedback controller. It also explains EKF in detail. All the equations responsible for ADCS algorithm are implemented in MSP-EXP432E401Y launchpad within a function named *main_sunpointing*(). This function is modified from the Delfi-n3Xt software as explained in Section 1.4, Chapter 1. It is important to note that matrix library as discussed in Chapter 3 is required to implement the ADCS on-board.

4.4 Advantages of using hardware in a loop approach

This section describes the advantages of using hardware in a loop approach as a design point in this thesis. In the Figure 4.1, control and estimation part of ADCS algorithm

[2] http://www.farnell.com/datasheets/1701911.pdf

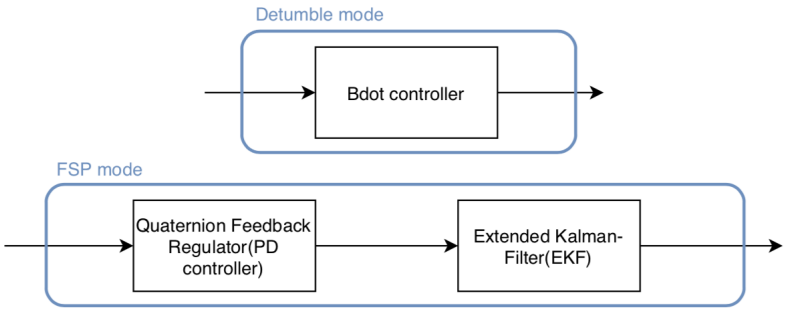


Figure 4.2: Estimation and Control modes of operation

was implemented as a part of on-board the Delfi-n3Xt software. But, in real case scenario, sensors perform the required measurement. And actuators control the satellite position and rotational rate. Disturbance torques and satellite dynamics are exerted by atmosphere around satellite (space environment). But, all these other models could be relatively modelled by considering nominal, non-nominal and practical parameters. This work has been claimed for robustness analysis by [1]. Hence, if we can utilize the MATLAB design that is developed in [1], we can eliminate the need for sensor and actuator, hardware, drivers and interface. Moreover, this also will eliminate, the need for creating a virtual test-bench for space atmosphere. Hence, a hardware in loop model as shown in Figure 4.1 is considered for design, experimentation and verification in this thesis.

4.5 Optimization

This section explains the optimization techniques used to improve performance of ADCS implemented in this thesis.

Firstly, optimization were done on DSP based arithmetic operations. The three DSP alternatives considered for this thesis are explained in Chapter 3. These implementations are considered in form of three alternatives. The DSP optimization done on all three alternatives are listed together in first subsection. These three implementations are compared in terms of performance and energy in Chapter 5.

Secondly, optimization are performed using compiler optimization options. The second subsection explains these compiler options in detail.

4.5.1 DSP and memory optimization

This subsection explains DSP and memory optimization done in this thesis. As explained above, three alternatives are implemented. In each of these alternatives, DSP and memory optimization were done as listed below:

- Redundant arithmetic were removed. For example multiplication by 2, followed by a, division by 2.

- There exists a factorial computation ($n!$) as part of Taylor series expansion, in order to perform exponential calculation. This is shown in Equation 4.1. To reduce computation load, $\frac{1}{n!}$ is replaced with a look-up table containing reciprocal of factorial. The source code is listed in (Appendix A.7).

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!} \tag{4.1}$$

- Division by a constant is replaced with multiplication by reciprocal of constant. For example, as stated above, the look-up includes the reciprocal of constant. This is multiplied with x^n (Equation-4.1) instead of division. The source code is listed in Appendix A.8.
- Multiplication with 2 were replaced with left shift once, as shown in Appendix A.9.
- Division with 2 were replaced with right shift once, as shown in Appendix A.10.
- Multiple accesses on same index of *sin*/*cos* look-up table, were replaced with single access, followed with a local variable storage, as shown in Appendix A.11.
- Multiple computations on same data were replaced with, single computation, followed with a local variable storage, as shown in Appendix A.12.
- IGRF reference look-up table was implemented using 3 dimensional integer array, instead of, matrix *struct* used by the Delfi-n3Xt matrix library. This saves memory usage for storing dimension of matrix. And, a const qualifier is used to indicate compiler to save the IGRF look-up in flash memory, as shown in Appendix A.13.

4.5.2 Compiler optimization flags

This subsection explains compiler optimization flags (-O0, -O1, -O2, -O3) that are used to study the trade-off between performance and code-size. A right candidate of optimization flag for implementation are concluded in Chapter 5. In order to reduce code size or improve performance of application ARM compiler provides various optimization level/flag. Primarily, one can optimize for performance or for code size [3]. However, there are several options for finer control of the optimization techniques. These optimization options are listed in subsection below.

4.5.2.1 -O0 Minimum optimization

This is the default and minimum optimization setting. Using this option one can achieve maximum debug capability as the generated code has a direct correlation of source code that is written. All optimization options that interfere with debug view are disabled. In other words using this option user can set breakpoints within reachable or dead code and value of variables are available everywhere within its scope. Hence, when the code is still being tested for functionality and debug capability is important this is one of two option to choose from (i.e., also -O1 is a candidate).

[3] `http://infocenter.arm.com/help/topic/com.arm.doc.100748_0606_00_en/compiler_user_guide_100748_0606_00_en.pdf`

4.5.2.2 -O1 Restricted optimization

This optimizes more for performance compared with -O0. This compiler option removes unused inline or static functions or variables. In other words, setting a break point in dead code is not possible. If the variable location is reused then content of these variables can no longer be accessed even in there scope. Functions that result in no side effects are removed. Hence, this optimization level can result in good correspondence between source code and object code. ARM recommends this option for debugging.

4.5.2.3 -O2 High optimization

This optimizes more for performance compared with -O1. I performs more aggressive instruction scheduling and can result in many to one mapping from object code to source code. The reported variable values and expected values might be different at any point as the instructions are allowed to cross sequence points. Hence, this optimization flag is not recommended for debugging, but for increase in performance.

4.5.2.4 -O3 Maximum optimization

This optimizes more for performance compared with -O2. This option generally result in poorer debug view even compared with -O2 and ARM recommends lower optimization level for this purpose . It includes High-level scalar optimization such as loop unrolling. This can result in a significant performance gain at a small code size cost, but at the risk of a longer build time. This regressive optimization can effectively rewrite the source code resulting in a smallest source-code to object-code density. Such a high level of optimization results in worst correlation from object code to source code and can hence result in a worst debug view. Since, this optimization affects the mapping of object code to source code there are additional optimization levels that can be chosen such as -Ospace and -Otime. These advanced compiler optimization are not utilized for this thesis work, curious readers are referred to [4]for more detailed explanation.

4.6 Summary

This chapter describes hardware in a loop simulation. It presents the two different modes of operation that are used in this thesis. It enlists different optimization techniques used in this thesis work. Primarily these were divided into two categories: DSP optimization and compiler optimization. It explained each of these optimization technique, in terms of, use case, in this thesis.

[4]http://infocenter.arm.com/help/topic/com.arm.doc.100748_0606_00_en/compiler_user_guide_100748_0606_00_en.pdf

5 Results

In this chapter, three different DSP alternatives are compared with respect to performance and energy, specifically for the ADCS used in the Delfi-n3Xt. These three different DSP alternatives have been presented in Chapter-3. The Delfi-n3Xt employs the use of DP implementation and this is considered as baseline. As described in Chapter-4 an MSP-EXP432E401Y launchpad is used with a hardware in the loop approach. However, by using this approach there is a considerable overhead due to the TCP/IP stack. But, in a real satellite software framework, this will not be present. Hence, a method that removes this overhead is considered. In general, the performance can be boosted using compiler optimization flags. As explained in Chapter 4 these flags result in a trade-off between performance and code-size. Hence, performance measurements for different compiler optimization flags and the resulting code-size are presented. It is important to note that the MSP-EXP432E401Y launchpad does not feature on-board energy trace. In order to perform power measurements, a traditional current measurement method is employed. This current measurements are then multiplied with voltage to produce the power measurements. A total energy consumption heuristic is proposed to extrapolate the energy measurements. Using this heuristic the total energy consumption for different DSP alternatives are calculated. It is important to observe that all three implementation alternatives must satisfy the functional requirements. Hence, this correctness is confirmed by means of graphs. Lastly, conclusions are drawn based on energy and performance measurements.

First this chapter begins with presenting performance versus code size trade-off for different compiler optimization level. This includes Compute Time (CT) for detumble and FSP mode separately. Secondly, power measurements are presented. Thirdly, the total energy is extrapolated using the proposed heuristic. Fourthly, graphs showing functional correctness are plotted. Finally, the chapter closes by drawing conclusions.

5.1 Performance verses code size trade-off for different compiler optimization levels

Chapter 4 presented performance verses code-size trade off provided by ARM. This understanding is henceforth, employed to study ADCS, specifically used in the Delfi-n3Xt. This section presents the CT for detumble and FSP mode separately. It tabulates code-size for each compiler optimization flag. The compiler flag that results in best performance is chosen to determine the performance boost with respect to baseline implementation. Then, performance boost are tabulated.

5.1.1 Performance gain in detumble and FSP mode

This subsection presents performance gain in detumble and FSP mode. The CT in seconds, recorded by detumble and FSP mode, using different compiler optimization flags are tabulated. As shown in Table 5.1 for FSP mode and Table 5.2 for detumble mode.

Table 5.1: CT in FSP mode, with different compiler optimization flag

DSP alternatives	-O0 (s)	-O1 (s)	-O2 (s)	-O3 (s)	-O4
DP	1157.15513	1120.66666	1117.55269	1159.96650	
SP	471.798210	440.436187	429.986995	430.712860	
FxP $< 32, 4 >$	427.337499	178.202572	169.925313	167.888228	163.528

Table 5.2: CT in detumble mode, with different compiler optimization flag

DSP alternatives	-O0 (s)	-O1 (s)	-O2 (s)	-O3 (s)
DP	0.79985517	0.79764839	0.76478392	0.82630401
SP	0.28138630	0.18191871	0.17492267	0.17576879
FxP $< 64, 36 >$	2.32246298	0.79596296	0.67871100	0.62388371

By using the Table 5.2 and 5.1 we can note that, O-2 compiler flag gives the best performance for the floating-point arithmetic(DP and SP), and O-3 gives the best performance for FxP arithmetic. Using these boosted options a speed-up is calculated, with respect to baseline. Let us consider Total Compute Time (TCT) as total compute time in FSP mode. This can be seen in Table 5.3 for detumble mode and Table 5.4 for FSP mode.

Table 5.3: Performance gain in detumble mode

DSP alternatives	CT	Speed-up
DP	0.76478392	-N.A-
SP	0.17492267	4.37x
FxP $< 64, 36 >$	0.62388371	1.22x

Table 5.4: Performance gain in FSP mode

DSP alternatives	TCT	Speed-up
DP	1117.55269	-N.A-
SP	429.986995	2.59x
FxP $< 32, 4 >$	163.528	6.83x

5.1.2 Code size in kB

This subsection tabulates the code size in kB, as the compiler optimization flag is varied. We can note from the datasheet of MSP-EXP432E401Y that, it has 1 MB of flash memory and 256 kB of SRAM [1]. As shown in Table 5.5 and Table 5.6, it can be noticed that the variation in code size, is not considerable.

Table 5.5: Flash memory usage, for different compiler optimization flag

DSP alternatives	-O0 (kB)	-O1 (kB)	-O2 (kB)	-O3 (kB)
DP	933.183	926.531	926.491	926.407
SP	930.719	923.595	922.691	922.591
FxP	927.587	923.903	923.904	922.579

Table 5.6: SRAM usage, for different compiler optimization flag

DSP alternatives	-O0 (kB)	-O1 (kB)	-O2(kB)	-O3 (kB)
DP	183.350	183.314	183.314	183.330
SP	181.362	181.334	181.334	181.346
FxP	181.671	181.641	181.643	183.410

5.2 Power measurements

A traditional current measurement method is used to perform power measurement. A multimeter in ammeter setting is connected in series between 3.3v line and micro controller. The current is recorded for three different DSP alternatives and bare-metal TCP/IP implementation, as show in Table 5.7. The Power equation, were power is given by product of voltage and current is used. As shown below:

$$P = V \times I \tag{5.1}$$

Where, V stands for voltage i.e., $3.3\,v$ in our case, and current I is the measurement recorded on ammeter. Using the above equation power is calculated and tabulated in Table 5.7.

Table 5.7: Power measurements in mW, for different DSP alternatives

DSP alternatives	Current (mA)	Power (mW)
Bare-metal TCP/IP	90.0	297.0
DP	106.2	350.46
SP	105.9	349.47
FxP $< 32, 4 >$	105.2	347.16

[1] http://www.ti.com/document-viewer/MSP432P401R/datasheet/specifications - t287270728

Bare-metal TCP/IP implementation is introduced for the first time in Table 5.7. Bare-metal TCP/IP implementation is made in order to remove the unwanted overhead that is added. To understand this let us look at Figure 1.2, Simulation of ADCS with hardware in a loop. Here we can see that TCP/IP is the way by which both MATLAB and MSP432 hardware, exchange data in order to form a closed-loop. Bare-metal TCP/IP implementation implements the similar closed loop, but without ADCS in MSP432, and without any models in MATLAB. The MSP432 will run a Bare-metal TCP/IP server implementation which will echo received data. Similarly, MATLAB will implement TCP/IP echo client. It is worth noting that if power from such an implementation is subtracted from actual hardware in loop implementation, as shown in Figure 1.2, then we arrive at the interesting part of the power which will actually be part of satellite software. In other words, this gives the power computation of actual ADCS algorithm. It is shown in Table 5.8.

Table 5.8: Power measurements in mW, for ADCS and DSP alternatives

DSP alternatives	Power(mW)
DP	53.46
SP	52.47
FxP $< 32, 4 >$	50.16

5.3 Total energy extrapolation

As mentioned in Chapter 2, the Control Step Time (CST) Δt=2 s for the Delfi-n3Xt. However, the whole 2 s time window is not used for computation. Hence, each CST can be divided into compute and hibernation time for each control step as shown in Figure 5.1.

Figure 5.1: Continuous control step windows

Let us consider Control Step Compute Time (CSCT) as compute time for each control step, then Control Step Hibernation Time (CSHT) can be given by:

$$\text{CSHT} = (\Delta t - \text{CSCT}) \tag{5.2}$$

The performance gain tabulation, Table 5.4 records TCT for the FSP mode. The total number of control steps in FSP mode for the performed simulation is 16960. In other words, Total Control Steps (TCS)=16960. Hence, CSCT can be calculated as:

$$\text{CSCT} = \frac{\text{TCT}}{\text{TCS}} = \frac{\text{TCT}}{16960} \tag{5.3}$$

The Table 5.8 gives the Compute Power (CP) for the different implementations. However, to calculate approximate Hibernation power (HP) we can refer to TI data-sheet. Using

the low power mode, LPM4 TI records current consumption to be as low as $2.2\mu A$ [2]. Using Equation 5.1 we can calculate HP= $2.2\mu A * 3.3v = 7.26\mu W$. The total energy can be computed using the above defined equations. The following heuristic is proposed for total energy calculation:

$$E = \text{TCS} \times ((\text{CSCT} \times \text{CP}) + (\text{CSHT} \times \text{HP})) \tag{5.4}$$

The Table-5.9 presents the energy extrapolation. Using the Equation 5.4 total energy (E) is calculated. Energy saving is calculated with respect to E of baseline (Which is 59.97939). The energy calculation heuristic that is proposed might not be the most optimal approach. But, this provides a closest intuitive approximation to energy consumption in a real satellite.

Table 5.9: Total energy extrapolation

DSP alternatives	TCT (s)	CSCT (s)	CSHT (s)	E (J)	Energy saving
DP	1117.55269	0.06589	1.93411	59.97939	-N.A-
SP	429.986995	0.02535	1.97465	22.79688	2.631x
FxP $< 32, 4 >$	167.888228	9.89907m	1.99010	8.66631	6.920x

5.4 Functional correctness

This subsection presents the graphs that confirm the functional correctness. As discussed earlier the two main functional requirements are:

- After detumble mode, for proper functioning of satellite, rotational rate of satellite must be below 1 $^\circ$ /s.
- In FSP mode, Sun-pointing error must not exceed 25 $^\circ$ for non eclipse period.

The following subsection will provide a summery about each plots. Each subsection will display plots for three different DSP alternatives.

5.4.1 Rotational rate in detumble mode

In this subsection, rotational rate plots are presented. It can be seen in Figure 5.3, Figure 5.2 and Figure 5.4. It is important to note that these plots verify the functional correctness of the first requirement about the rotational rate. That is close to $2 * 10^4$ on the x-axis the rotational rate falls below 1 $^\circ$ /s.

5.4.2 Sun-pointing error

This subsection presents plots for Sun-pointing error. It is important to note that these plots satisfy the second functional requirement. That is the maximum pointing error is below 25 $^\circ$. This can be seen from Figure 5.5, Figure 5.6 and Figure 5.7.

[2]http://www.ti.com/document-viewer/MSP432P401R/datasheet/specifications - t287270728

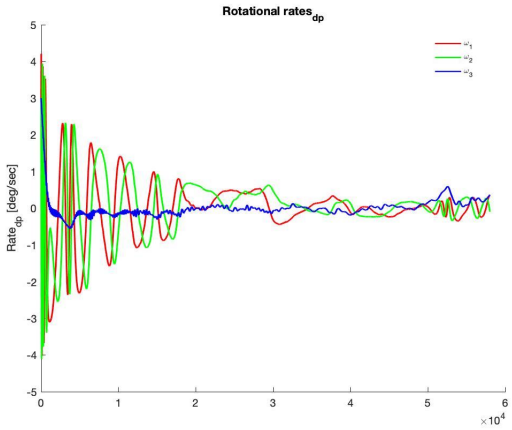


Figure 5.2: Rotational rate in detumble mode DP

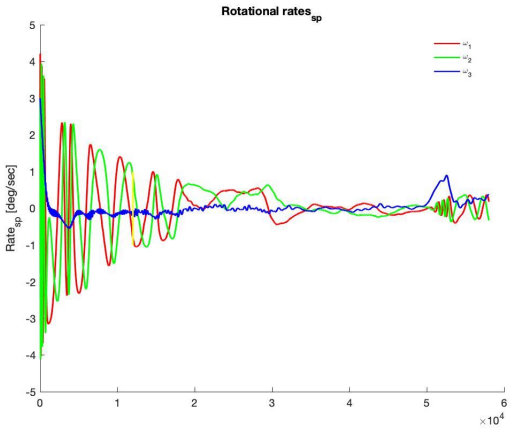


Figure 5.3: Rotational rate in detumble mode SP

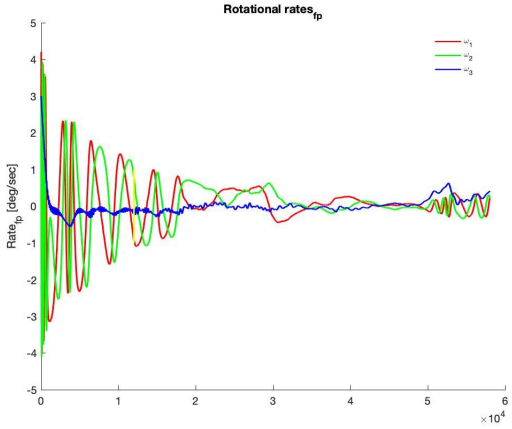


Figure 5.4: Rotational rate in detumble mode FxP

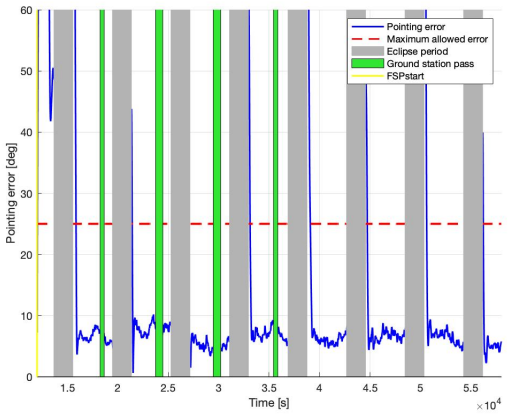


Figure 5.5: Sun-pointing error for DP

5.4.3 Rotational rate error in FSP mode

This subsection presents plots for rotational rate in FSP mode. These plots also confirm the first functional requirement, that is after detumble mode, the rotational rate error must be below 1 $^\circ$ /s. This can be seen from Figure 5.8, Figure 5.9 and Figure 5.10.

5.4.4 Quaternion plots

This subsection presents quaternion trend plots. It has three subplots. First, the true quaternion that is expected estimated quaternion, from MATLAB model. Second is the estimated quaternion ($\hat{\boldsymbol{q}}$), from attitude estimation algorithm. Third, is the desired quaternion ($\boldsymbol{q}_d$) that are fixed at $\boldsymbol{q}_0$ the initial value stated in Chapter 2. This can be seen in Figure 5.11, Figure 5.12 and Figure 5.13.

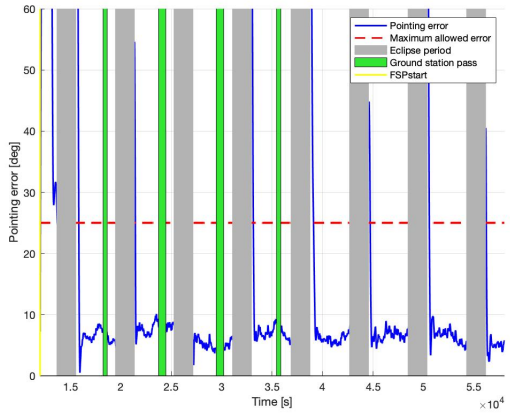


Figure 5.6: Sun-pointing error for SP

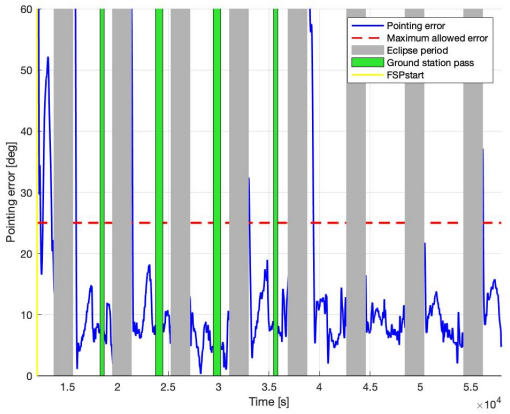


Figure 5.7: Sun-pointing error for FxP

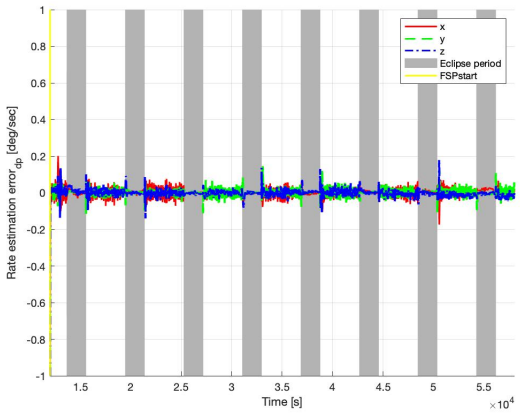


Figure 5.8: Rotational rate error in DP

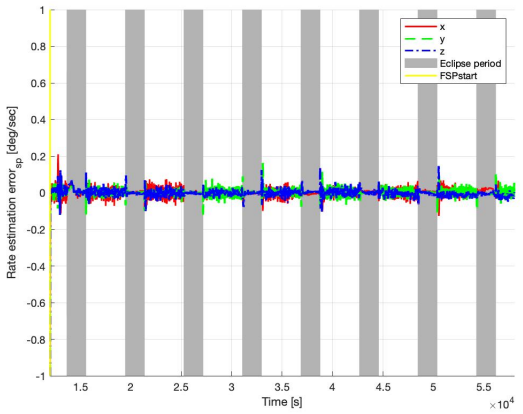


Figure 5.9: Rotational rate error in SP

5.4.5 Angular velocity plots

This subsection presents angular velocity trend plots. It has three subplots. First, the true angular velocity that is expected estimated angular velocity, from MATLAB model. Second is the estimated angular velocity ($\hat{\boldsymbol{\omega}}$), from attitude estimation algorithm. Third, is the desired angular velocity ($\boldsymbol{\omega}_d$) that are fixed at $\boldsymbol{\omega}_0$ the initial value stated in Chapter 2. This can be seen in Figure 5.14, Figure 5.15 and Figure 5.16. In addition, Figure 5.17 shows coastal line of earth with the Delfi-n3Xt orbiting.

5.5 Conclusion

Based on the experimentation's and study done in this chapter, the following conclusions can be drawn:

- The SP based implementation is 2.59 times faster and consumes 2.631 less energy

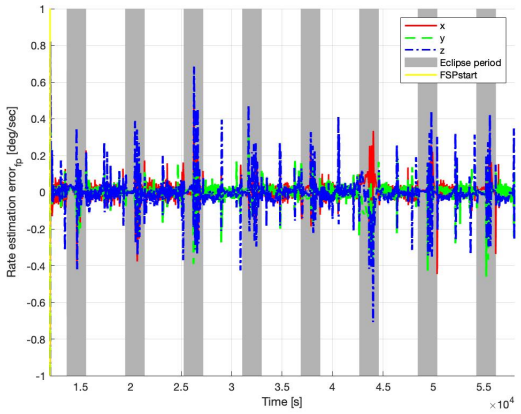


Figure 5.10: Rotational rate error in FxP

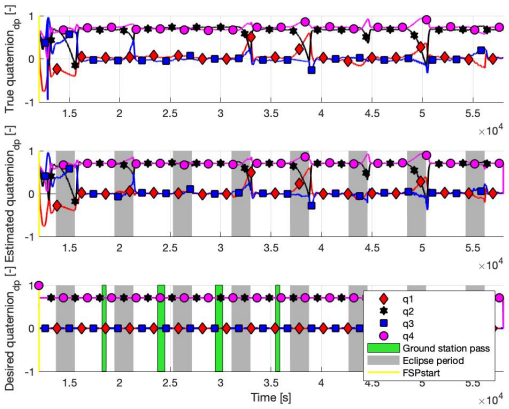


Figure 5.11: Quaternion plots in DP

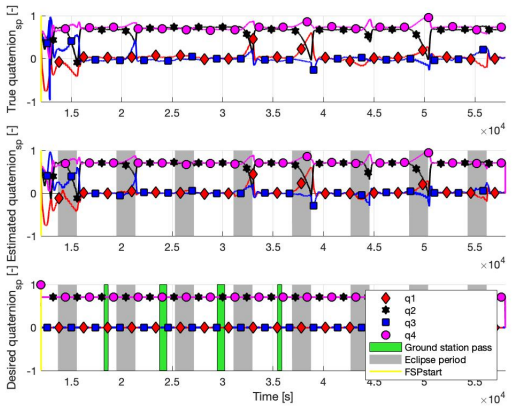


Figure 5.12: Quaternion plots in SP

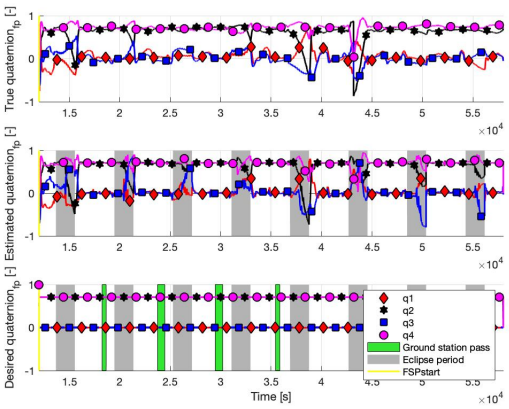


Figure 5.13: Quaternion plots in FxP

than the baseline.

- The FxP based implementation is 6.66 times faster and consumes 6.920 less energy than the baseline.
- From the functional correctness plots, it can be verified that SP provides acceptable performance and energy consumption. Moreover, ADCS for the Delfi-n3Xt does not require DP arithmetic.
- The FxP based implementation has more error compared to SP, but satisfies functional requirements. This can be seen from sun-pointing error Figure 5.7. In future satellite projects, if more accuracy is expected, then SP based implementation is proposed over FxP.
- There is no considerable changes observed in code size with respect to compiler optimization flags. Hence, we may focus on performance gain and neglect the

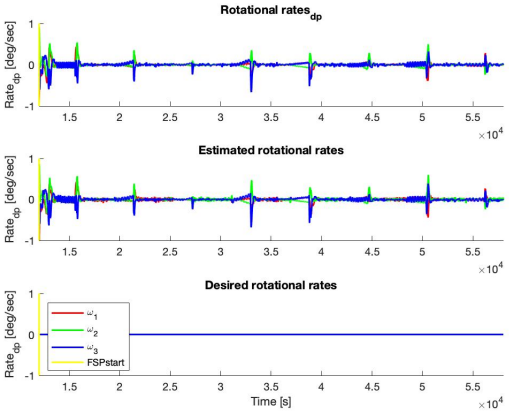


Figure 5.14: Angular velocity in DP

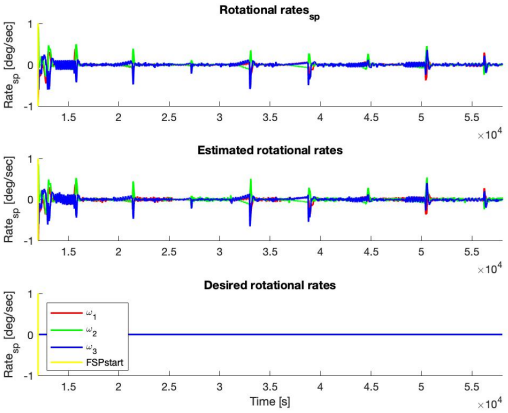


Figure 5.15: Angular velocity in SP

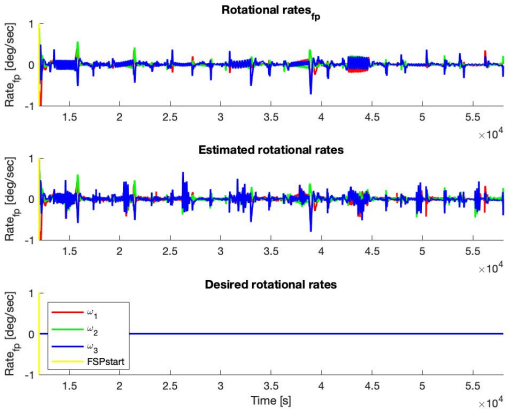


Figure 5.16: Angular velocity in FxP

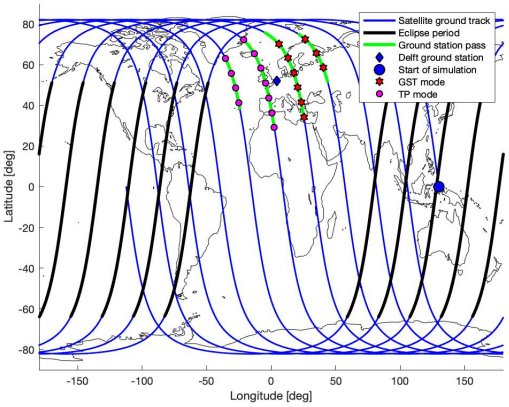


Figure 5.17: The Delfi-n3Xt orbit

consideration on code size.

6 Conclusions

This chapter presents conclusions of this work. Section 6.1 summarizes the thesis. Section 6.2 presents conclusions drawn. Section 6.3 enlists main contributions from this thesis. Section 6.4 presents recommendations on possible future work.

6.1 Summary

This section offers a summary for the whole thesis. Furthermore, it presents a short description in terms of each chapter.

Chapter 1 provides an introduction to the thesis. Firstly, it presents motivation and objective for this work. Secondly, it gives an overview of research methodology that has been used throughout this thesis. Lastly, it enlists personal contribution and outlines the thesis.

Chapter 2 explains ADCS algorithm that has been used by the Delfi-N3xt satellite, and this thesis. Firstly, it describes an overview of reference frames. Secondly, it defines rotational kinematics and three main attitude representations. Thirdly, it describes a model of physical world that is required in order to understand the Sun direction and Earth's magnetic field calculation, performed in the Delfi-N3xt. Fourthly, it explains attitude estimation algorithm, i.e., EKF algorithms. Fifthly, it describes an overview of attitude control algorithms used by the Delfi-N3xt. Lastly, it presents power budget for ADCS, in the Delfi-N3xt.

Chapter 3 explains three different DSP alternatives that have been used in this thesis. Firstly, it explains two floating-point representations that are used in this thesis. Secondly, it explains FxP representation and arithmetic techniques. Lastly, it presents an overview for extension of the Delfi-N3xt FxP library.

Chapter 4 explains the hardware in a loop simulation with respect to this thesis. Firstly, it briefly introduces the TCP/IP protocol. Secondly, it describes the implementation of TCP/IP server and client. Thirdly, it presents the two different modes of operation that have been used in this thesis. Lastly, It enlists different optimization techniques used in this thesis work. Primarily these were divided into two categories: DSP optimization and compiler optimization.

Chapter 5 compares different DSP alternatives, presented in Chapter 3, with respect to performance, energy and code-size. Firstly, it presents performance versus code size trade-off for different compiler optimization levels. Secondly, it enlists power measurements. Thirdly, the total energy is extrapolated using the proposed heuristic. Fourthly, graphs showing functional correctness are plotted. Lastly, the chapter closes with drawing conclusions.

Chapter 6, the current chapter summarizes the whole thesis. Firstly, it begins with a summary of the whole thesis document. Secondly, it enlists main contributions of this

thesis. Lastly, it proposes the recommendations for future work.

6.2 Conclusions

This section underlines the conclusions that can be drawn from this thesis.

1. ADCS is a critical subsystem. It was important to satisfy two main objectives, before proposing energy improvement alternatives. This thesis considers the following two main objectives of ADCS:
   - Detumble mode, to reduce angular velocity of the satellite below 1 $^{\circ}$/s.
   - FSP mode, to point solar panels towards the Sun, with a maximum allowed Sun pointing error of 25 $^{\circ}$.
2. The FxP implementation of ADCS is proposed in this thesis. This performs $\approx$ 6.7 times faster than the baseline. In addition it consumes $\approx$ 7 times less energy, with respect to the baseline.
3. The FxP matrix library implements all the functions necessary for the Delfi-N3xt ADCS. Therefore, this library could be directly, or with a little modification, be used for future satellite projects.
4. The power budget for the Delfi-N3xt ADCS software is $\approx$ 10 % of total nominal power budget. A major amount of ADCS power budget is consumed by sensors and actuators. Hence, the performance and energy gain that are observed in this thesis are not considerable to improve energy, for direct deployment, such as, in terms of the Delfi-N3xt total nominal power budget. It is important to note that the only computationally intensive algorithm present in the Delfi-N3xt was ADCS.
5. However, in future, if there exists more compute intensive algorithm, that might require considerable amount of total power, then the conclusions observed in this thesis can be used as an initial study. But, this does not mean that the same conclusions can be drawn for future satellite algorithm. Since, it must also satisfy the precision requirements of such algorithm, which directs towards use of a similar research methodology, to define meaningful conclusions.
6. This thesis extends the Delfi-N3xt ADCS algorithm with two modes of operation. This implementation uses DP alternative, and acts as a baseline. For the ADCS, implemented in this thesis, SP performs $\approx$ 3 times better and saves $\approx$ 2.7 times more energy, compared to the baseline.
7. Moreover, this study also suggests that SP has a similar accuracy compared with DP implementation. Hence, for future implementation this thesis suggests to consider SP as a better alternative than DP, which was used in the Delfi-N3xt.
8. Lastly, conclusions drawn in this thesis, in terms of power and energy consumption, are considerably important for Delfi satellite project. Because, this thesis for the first time fully implements one of the 4 advanced modes. These advanced modes were missing in previous work by Johannes P.J. Reijneveld, M. Vos, et al.

6.3 Main contributions

The following are the main contributions from this thesis:

1. The Delfi-N3xt ADCS algorithm is extended with two modes of operation. This includes a complete implementation of FSP mode. This implementation uses DP alternative, and is used as a baseline throughout this work.

2. Porting of ADCS algorithm as mentioned above, to SP and FxP. It includes implementing a matrix library in, SP and FxP.

3. A MATLAB script that can auto-generate IGRF look-up table (header file), for both SP and FxP.

4. A proposed implementation in FxP that can perform ≈ 6.7 times faster than the baseline, and consumes ≈ 7 times less energy, than the baseline.

6.4 Future work

The following recommendation are made for possible future work:

- The other advanced modes of the Delfi-N3xt use the same EKF algorithm. With a minimum effort it must be possible to extend current implementation with other advanced modes. Hence, it is recommended to extend the implementation by complete the Delfi-N3xt ADCS algorithm.

- In this thesis no changes were made with the gain parameters, compared with the Delfi-N3xt ADCS algorithm. By tuning derivative gain of the Quaternion feedback regulator controller, it is possible to tune the damping that is observed in FxP implementation. Tuning this parameter, might result in improved FxP accuracy.

- The current work uses hardware in a loop approach. However, it is recommended to interface sensor and actuators with the future implementation. This will give an accurate measurement of accuracy and performance achieved by FxP implementation.

- For simplicity, all the implementations in this thesis were done on a microcontroller featuring ARM-M4F. But, the area and power consumption of M3 is less compared with M4 [1]. This can result in power saving with FxP implementation. Hence, it is recommended to use a ARM-M3 processor and perform accurate measurements of power.

- By *gprof* based flat profiling of software developed in this thesis, it was observed that EKF propagation function consumes 34 % of total compute time. Hence, this can be accelerated using microcontroller featuring an accelerator. This might result in lowered energy consumption.

[1] https://www.arm.com/products/silicon-ip-cpu

- Results from a study shown in Figure 3.3 suggests that WL of 36 results in lowest error in FxP alternative. However, microcontrollers are designed for instance, to use a WL of 32 or 64 bit. Use of custom hardware implementation, using an FPGA, could benefit from such study. A custom WL could be considered for such implementation. In addition, software implementations from different satellite subsystems could be combined in a single FPGA. Moreover, sensor and actuator interface drivers could also be implemented together on the a same FPGA. This could result in an FPGA implementation higher performance and lower energy consumption, than the proposed alternative in this thesis.

Bibliography

[1] J. P. Reijneveld, "Design of the attitude determination and control algorithms for the delfi-n3xt," Masters thesis Space Systems Engineering, Tu Delft, January 2012.

[2] Q. Chu, "Spacecraft dynamics and control," Lecture Notes, Tu Delft, May 2018.

[3] J. Diebel, "Representing attitude: Euler angles, unit quaternions, and rotation vectors," *Matrix*, vol. 58, no. 15-16, pp. 1–35, 2006. [Online]. Available: https://www.astro.rug.nl/software/kapteyn/_downloads/attitude.pdf

[4] J. B. R. Teuling, "Top level design of the electrical power subsystem," Department of Space Systems Engineering, Delft University of Technology, Kluyverweg 1, 8th floor, 2629 HS Delft, Tech. Rep. DNX-TUD-TN-0019, July 2009, ©2010. All rights reserved. Disclosure to third parties of this document or any part thereof, or the use of any information contained therein for purposes other than provided for by this document, is not permitted, except prior and express written permission of Delft University of Technology.

[5] M. Haghayegh, "Design, implementation and verification of the attitude determination and control algorithm for the delffi satellite," Masters thesis Faculty of Aerospace Engineering, Tu Delft, July 2015.

[6] J. Guo, J. Bouwmeester, and E. Gill, "In-orbit results of Delfi-n3Xt: Lessons learned and move forward," *Acta Astronautica*, 2016.

[7] J. A. Rodriguez, "Estimation: Kalman filters for attitude and orbit determination," Masters thesis Space Systems Engineering, Tu Delft, May 2011.

[8] M. Vos, "Delfi-n3xt's attitude determiniation and control subsystem implementation and verification of the hardware and software," Masters thesis Computer Engineering, Tu Delft, May 2013.

[9] S. Speretta, T. Soriano, J. Bouwmeester, J. Carvajal-Godinez, A. Menicucci, T. Watts, P. Sundaramoorthy, J. Guo, and E. Gill, "CubeSats to Pocketqubes: Opportunities and challenges," *Proceedings of the International Astronautical Congress, IAC*, 2016.

[10] S. Uludag, S. Speretta, E. Gill, J. Bouwmeester, and T. Pérez Soriano, *A New Electrical Power System Architecture For Delfi-PQ*. Univelt Inc., 2017, vol. 163, pp. 511–522.

[11] E. Thébault, C. C. Finlay, C. D. Beggan, P. Alken, J. Aubert, O. Barrois, F. Bertrand, T. Bondar, A. Boness, L. Brocco, E. Canet, A. Chambodut, A. Chulliat, P. Coïsson, F. Civet, A. Du, A. Fournier, I. Fratter, N. Gillet, B. Hamilton, M. Hamoudi, G. Hulot, T. Jager, M. Korte, W. Kuang, X. Lalanne, B. Langlais, J. M. Léger, V. Lesur, F. J. Lowes, S. Macmillan, M. Mandea, C. Manoj,

S. Maus, N. Olsen, V. Petrov, V. Ridley, M. Rother, T. J. Sabaka, D. Saturnino, R. Schachtschneider, O. Sirol, A. Tangborn, A. Thomson, L. Tøffner-Clausen, P. Vigneron, I. Wardinski, and T. Zvereva, "International geomagnetic reference field: The 12th generation international geomagnetic reference field - The twelfth generation," *Earth, Planets and Space*, 2015.

[12] A. C. Stickler and K. Alfriend, "Elementary Magnetic Attitude Control System," *Journal of Spacecraft and Rockets*, 1976.

[13] B. Wie, H. Weiss, and A. Arapostathis, "Quarternion feedback regulator for spacecraft eigenaxis rotations," *Journal of Guidance, Control, and Dynamics*, 1989.

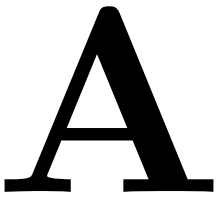
[14] J. . S. W. um, Ki-Il & Kang, "An optimizing floating-point to integer c program converter for fixed-point digital signal processors.l," in *AUTOSCALER for C,(DATE) 2000*, 2010, pp. 1 –6.

[15] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*. New York, NY, USA: Oxford University Press, Inc., 2000.

[16] C. Kozierok, *The TCP/IP Guide: A Comprehensive, Illustrated Internet Protocols Reference*. San Francisco, CA, USA: No Starch Press, 2005.

[17] T. Instruments, *TI_RTOS User Guide*, TI.

Fixed-point Design definition



A.1 Floating point to FxP

```
#define fixedpt_rconst(R) ((fixedpt)((R) * (((fixedptd)1 << FIXEDPT_FBITS) +
    ((R) >= 0 ? 0.5 : -0.5))))
```

A.2 Conversion from one FWL to other

```
#define fixedpt_cov_F28_to_F22(R) ((R>>6))
```

A.3 Defining a FxP Variable

```
#define FIXEDPT_PI fixedpt_rconst(3.14159265358979323846)
```

A.4 Multiplication

```
/* Multiplies two fixedpt numbers, returns the result. */
static inline fixedpt
fixedpt_mul(fixedpt A, fixedpt B)
{
    int64_t product;
    product = (int64_t)A * B;

    return (fixedpt)(rnd(product >> (FIXEDPT_FBITS-1)));

}
```

A.5 Rounding result from Multiplication

```
#define LOWER_RND               0X00000001
static inline int32_t rnd(int64_t x)
{
    int32_t y;
```

```
    y=x>>1;
    if(x&LOWER_RND)
        y=y+1;

    return(y);
}
```

A.6 Division

```
/* Divides two fixedpt numbers, returns the result. */
static inline fixedpt
fixedpt_div(fixedpt A, fixedpt B)
{
    if(B==0)
        return 0;
    else
        return (((fixedptd)A << FIXEDPT_FBITS) / (fixedptd)B);
}
```

A.7 Look-up Factorial

```
        static int32_t k[17]={FIXEDPT_ONE,FIXEDPT_ONE,fixedpt_rconst(1.0/2),
       fixedpt_rconst(1.0/6),fixedpt_rconst(1.0/24),
       fixedpt_rconst(1.0/120),fixedpt_rconst(1.0/720),
       fixedpt_rconst(1.0/5040),fixedpt_rconst(1.0/40320),
       fixedpt_rconst(1.0/362880),fixedpt_rconst(1.0/3628800),
       fixedpt_rconst(1.0/39916800), fixedpt_rconst(1.0/479001600),
       fixedpt_rconst(1.0/1932053504), fixedpt_rconst(1.0/1278945280),
       fixedpt_rconst(1.0/2004310016), fixedpt_rconst(1.0/2004189184)};
```

A.8 Exponential calculation

```
    #define MAT16_EXP(A, terms, RES) do { \
      MAT16_int32_t_t temp1, temp2, temp3; \
      MAT_PREP_DIAG((RES), (A).rows, (A).cols, FIXEDPT_ONE); \
      MAT_PREP_DIAG(temp1, (A).rows, (A).cols, FIXEDPT_ONE); \
      uint8_t i;\
      for (i=1; i <= (terms); i++){ \
         MAT_MUL((A), temp1, temp2); \
         temp1 = temp2; \
         MAT_SCALE(temp2, (int32_t)k[i], temp3);\
            MAT_ADD((RES), temp3, (RES)); \
      } \
```

```
} while(0)
```

A.9 Multiplication with two

```
M = ((mul_alg_F22(GC_PI , (fixedpt_rconst_64_F22(B_meas ->
    spt_begin+t_error))))<<1); //mul with 2 is left shift once
```

A.10 Division with two

```
E_32=E_32>>1;//divide by 2 is right shift once
```

A.11 Multiple access replaced with single access

```
cosr = cos_F31(RAAN); // this is fixed why access again and again?
sinr = sin_F31(RAAN);
coso = cos_F31(omega);
sino = sin_F31(omega);
cosi = cos_F31(incl);
sini = sin_F31(incl);

l1 = (fixedpt_mul_F31(cosr , coso) -
    fixedpt_mul_F31((fixedpt_mul_F31(sinr,sino)) , cosi));
l2 = (fixedpt_mul_F31(-cosr , sino) -
    fixedpt_mul_F31((fixedpt_mul_F31(sinr,coso)) , cosi));

m1 = (fixedpt_mul_F31(sinr , coso) +
    fixedpt_mul_F31((fixedpt_mul_F31(cosr , sino)) , cosi));
m2 = (fixedpt_mul_F31(-sinr , sino) +
    fixedpt_mul_F31((fixedpt_mul_F31(cosr , coso)) , cosi));

n1 = fixedpt_mul_F31(sino , sini);
n2 = fixedpt_mul_F31(coso , sini);
```

A.12 Multiple computation replaced with single computation

```
local_az = (((*az)>>18) + 180);
local_elev = (((*elev)>>18)+90);
B_E_ref.comp[0] = B_ref_lut[local_az][local_elev][0];
B_E_ref.comp[1] = B_ref_lut[local_az][local_elev][1];
```

```
        B_E_ref.comp[2] = B_ref_lut[local_az][local_elev][2];
```

A.13 IGRF look-up in fixed-point$< 32, 4 >$

```
const int32_t B_ref_lut[361][181][3]={
    {
    {2558,-1656,10707},
    {2649,-1670,10816},
    {2739,-1684,10920},
    {2827,-1697,11019},
    {2914,-1710,11114},
    {3000,-1723,11203},
    {3085,-1735,11288},
    {3168,-1746,11366},
    {3249,-1758,11439},
    {3329,-1769,11506},
    {3408,-1779,11567},
    {3484,-1789,11622},
    {3559,-1798,11671},
    {3633,-1807,11713},
    {3704,-1815,11749},
    {3774,-1822,11778},
    {3842,-1829,11801},
    {3908,-1835,11818},
    {3973,-1840,11828},
    {4036,-1845,11832},
    {4096,-1850,11829},
    {4155,-1853,11820},
    {4212,-1856,11805},
    {4267,-1858,11783},
    {4320,-1860,11756},
    {4371,-1861,11724},
    {4420,-1861,11686},
    {4467,-1861,11642},
    {4511,-1860,11593},
    {4554,-1859,11540},
    {4595,-1856,11482},
    {4633,-1854,11420},
    {4669,-1851,11353},
    {4702,-1847,11283},
    {4733,-1843,11209},
    {4762,-1838,11132},
    {4788,-1832,11051},
    {4811,-1826,10968},
    {4831,-1821,10883},
    {4848,-1814,10795},
    {4863,-1806,10705},
    {4874,-1799,10613},
```

```
    {4882,-1790,10520},
    {4887,-1781,10425},
    {4887,-1772,10329},
    {4885,-1763,10232},
    {4879,-1753,10135},
    {4868,-1743,10037},
    {4854,-1732,9938},
    {4836,-1721,9839},
    {4814,-1709,9741},
    {4787,-1698,9642},
    {4757,-1686,9544},
    {4721,-1674,9446},
    {4682,-1661,9349},
    {4637,-1648,9251},
    {4589,-1635,9155},
    {4535,-1622,9060},
    {4477,-1608,8966},
    {4415,-1595,8872},
    {4347,-1581,8781},
    {4275,-1567,8689},
    {4198,-1553,8600},
    {4117,-1538,8511},
    {4031,-1524,8424},
    {3941,-1509,8338},
    {3847,-1494,8254},
    {3748,-1479,8171},
    {3645,-1464,8090},
    {3538,-1449,8011},
    {3427,-1434,7933},
    {3313,-1419,7857},
    {3195,-1404,7782},
    {3075,-1388,7710},
    {2951,-1373,7639},
    {2824,-1358,7570},
    {2695,-1343,7503},
    {2564,-1328,7439},
    {2430,-1313,7376},
    {2295,-1298,7315},
    {2159,-1282,7257},
    {2021,-1267,7201},
    {1883,-1253,7147},
    {1744,-1237,7095},
    {1605,-1222,7046},
    {1466,-1208,6999},
    .
    .
    }
};
```

IGRF

B

B.1 MATLAB script to generate floating point header

```
%script to create look-up-table header file for B_ref
%az range -180 to +180 degree
%elev range -90 to +90 degree
magmode = 'IGRF';        % mode: dipole, IGRF, wrldmagm

fileID = fopen('dump/B_ref.h','w');
fprintf(fileID,'// B_Ref.h\n');
fprintf(fileID,'// Auto-written by matlab for igrf look-up\n');
fprintf(fileID,'// IGRF reference look-up dump for r fixed at 6978\n//
    B_E[azimuth][elevation]\n');
fprintf(fileID,'// note azimuth real range is from -180 to +180 degree but in
    LUT it has offset of +180\n');
fprintf(fileID,'// example azimuth = 0 in LUT is -180 in reality\n');
fprintf(fileID,'// similarly elevation has an offset of +90\n');
fprintf(fileID,'// Created by hemanth singh jagadeeshwar on 5/8/18.\n');
    fprintf(fileID,'const float B_ref_lut[361][181][3]={');
    fprintf(fileID,'\n');
for az_deg=-180:180
    fprintf(fileID,'{');
        fprintf(fileID,'\n');
    for elev_deg= -90:90
        r =6978;
        az=deg2rad(az_deg);
        elev=deg2rad(elev_deg);
        [r_Ecd(1),r_Ecd(2),r_Ecd(3)] = sph2cart(az,elev,r);
        B_E = getMagField(r_Ecd,magmode);
        B_E = round(B_E/2e-9)*2e-9;

        fprintf(fileID,'{%.20f,',B_E(1));
        fprintf(fileID,'%.20f,',B_E(2));
        fprintf(fileID,'%.20f}',B_E(3));

        if elev_deg ~= 90
            fprintf(fileID,',');
        end
                fprintf(fileID,'\n');

    end
            fprintf(fileID,'}');
            if az_deg ~= 180
```

```
                fprintf(fileID,',');
            end
                        fprintf(fileID,'\n');

end
    fprintf(fileID,'};');

fclose(fileID);
```

```
function [B_E] = getMagField(r_E,varargin)
% This function returns the magnetic field in ECEF at the given position
% r_E in km in ECEF

%% Convert the ECEF position to spherical coordinates
r_E = r_E*1000;                         % first a conversion to meters
[elong,nlat,r] = cart2sph(r_E(1),r_E(2),r_E(3));

%% Check varargin
if isempty(varargin)
    varargin{1} = 'dipole';
end
%% Select the right model
switch varargin{1}
    case 'dipole'
        mu_0 = 4*pi*10^-7; % magnetic permeability of free space [kg m A^-2
            s^-2]
        m = 7.94*10^22;   % Earth's best fit dipole moment [A m^2]
        B0 = mu_0*m/(4*pi*r^3);
        B_r = 2*B0*cos(.5*pi-nlat);
        B_t = B0*sin(.5*pi-nlat);
        B_E(1,1) = (B_r*cos(nlat)-B_t*sin(nlat))*cos(elong);
        B_E(2,1) = (B_r*cos(nlat)-B_t*sin(nlat))*sin(elong);
        B_E(3,1) = B_r*sin(nlat)+B_t*cos(nlat);
    case 'IGRF'

        B_tpr = igrf11syn(2010,r/1000,rad2deg(nlat),rad2deg(elong))*10^-9;
        B_rpt = [B_tpr(3);B_tpr(2);B_tpr(1)];
        C = get3RotMatrix([nlat;-elong;0],'231');
        B_E = C*B_rpt;
    case 'wrldmagm'
        B_E = wrldmagm(r/1000-6371.2, nlat, elong, 2010)*10^-9;
    otherwise
        warning('Unexpected Magnetic model. Nothing returned...');
end

end
```

B.2 MATLAB script to generate fixed-point header

```
%script to create look-up-table header file for B_ref
%az range -180 to +180 degree
%elev range -90 to +90 degree
fileID = fopen('dump/B_ref.h','w');
fprintf(fileID,'// B_Ref.h\n');
fprintf(fileID,'// Auto-written by matlab for igrf look-up\n');
fprintf(fileID,'// IGRF reference look-up dump for r fixed at 6978\n//
    B_E[azimuth][elevation]\n');
fprintf(fileID,'// note azimuth real range is from -180 to +180 degree but in
    LUT it has offset of +180\n');
fprintf(fileID,'// example azimuth = 0 in LUT is -180 in reality\n');
fprintf(fileID,'// similarly elevation has an offset of +90\n');
fprintf(fileID,'// Created by hemanth singh jagadeeshwar on 5/8/18.\n');
    fprintf(fileID,'const int32_t B_ref_lut[361][181][3]={');
    fprintf(fileID,'\n');
for az_deg=-180:180
    fprintf(fileID,'{');
        fprintf(fileID,'\n');
    for elev_deg= -90:90
        r =6978;
        az=deg2rad(az_deg);
        elev=deg2rad(elev_deg);
        [r_Ecd(1),r_Ecd(2),r_Ecd(3)] = sph2cart(az,elev,r);
        B_E = getMagField(r_Ecd,magmode);
        B_E = round(B_E/2e-9)*2e-9;

        fprintf(fileID,'{%d,',fixedpt_rconst(B_E(1)));
        fprintf(fileID,'%d,',fixedpt_rconst(B_E(2)));
        fprintf(fileID,'%d}',fixedpt_rconst(B_E(3)));

        if elev_deg ~= 90
            fprintf(fileID,',');
        end
                fprintf(fileID,'\n');

    end
            fprintf(fileID,'}');
            if az_deg ~= 180
                    fprintf(fileID,',');
            end
            fprintf(fileID,'\n');
end
    fprintf(fileID,'};');

fclose(fileID);
```

```
function [B_E] = getMagField(r_E,varargin)
```

```
% This function returns the magnetic field in ECEF at the given position
% r_E in km in ECEF

%% Convert the ECEF position to spherical coordinates
r_E = r_E*1000;                          % first a conversion to meters
[elong,nlat,r] = cart2sph(r_E(1),r_E(2),r_E(3));

%% Check varargin
if isempty(varargin)
    varargin{1} = 'dipole';
end
%% Select the right model
switch varargin{1}
    case 'dipole'
        mu_0 = 4*pi*10^-7; % magnetic permeability of free space [kg m A^-2
            s^-2]
        m = 7.94*10^22;   % Earth's best fit dipole moment [A m^2]
        B0 = mu_0*m/(4*pi*r^3);
        B_r = 2*B0*cos(.5*pi-nlat);
        B_t = B0*sin(.5*pi-nlat);
        B_E(1,1) = (B_r*cos(nlat)-B_t*sin(nlat))*cos(elong);
        B_E(2,1) = (B_r*cos(nlat)-B_t*sin(nlat))*sin(elong);
        B_E(3,1) = B_r*sin(nlat)+B_t*cos(nlat);
    case 'IGRF'

        B_tpr = igrf11syn(2010,r/1000,rad2deg(nlat),rad2deg(elong))*10^-9;
        B_rpt = [B_tpr(3);B_tpr(2);B_tpr(1)];
        C = get3RotMatrix([nlat;-elong;0],'231');
        B_E = C*B_rpt;
    case 'wrldmagm'
        B_E = wrldmagm(r/1000-6371.2, nlat, elong, 2010)*10^-9;
    otherwise
        warning('Unexpected Magnetic model. Nothing returned...');
end

end
```

```
function [outputArg1] = fixedpt_rconst(inputArg1)
%fixedpt_rconst Summary of this function goes here
%converts from double to fixed point F28(4.28)
if (inputArg1) >= 0
   internal = bitsll(1,28) + 0.5;
else
   internal = bitsll(1,28) - 0.5;
end
outputArg1= int32((inputArg1) * internal);
```