Problem Description

The NTNU Test Satellite project was started in September 2010, the goal is to build a launch ready double CubeSat. This thesis aims to further develop the Attitude Determination and Control System to a near flight ready version.

- Simulate the satellite with the previously chosen pointing and detumbling controllers to see how they perform with the current actuator and software design. Retune controllers if necessary.
- Simulate the pointing controller when there are noise on the angular velocity measurements to check robustness.
- Design, evaluate and simulate a controller to increase the angular velocity of the satellite, a tumbling controller.
- Design a second Attitude Determination and Control System hardware prototype. The second prototype will be based on the first prototype, but with additional sensors added for redundancy.
- Design the overall software architecture for the Attitude Determination and Control System.
- Find and implement an algorithm to calculate the best value for the data provided by the sensors that have hardware redundancy.
- Perform a failure analysis for the Attitude Determination and Control System to show how it might fail and how it can be avoided.

To be handed in by: 13.07.2015

Supervisor 1: Professor Jan Tommy Gravdahl

Supervisor 2: Roger Birkeland

Abstract

The NTNU Test Satellite is a student satellite program at NTNU that is part of the Norwegian Student Satellite Program run by the Norwegian Center for Space Related Education. The goal of the project is to send a student built and designed CubeSat to space. This thesis continues the work of developing a Attitude Determination and Control System for the NTNU Test Satellite.

In this thesis simulations have been run for two previously chosen controllers (detumbling and pointing) to ensure that the controllers will work with the hardware and software design of the Attitude Determination and Control System. A third controller has been developed to increase the angular velocity of the satellite, called a tumbling controller, and simulations have been performed to check its performance. Simulations have also been performed to check how the pointing controller works when there are noise on the angular velocity measurements. The simulations shows that all of the controllers will work satisfactory. Instability evidence is also presented for the tumbling controller.

A second prototype for the Attitude Determination and Control System hardware has been designed with a focus on redundancy, and the overall software architecture has been designed. Most of the software have been tested on a evaluation kit for the chosen microcontroller. A median voter has been implemented to handle the sensor redundancy in software. Lastly a simple failure analysis have been performed, fault trees have been used to show how the Attitude Determination and Control System might fail to perform as designed. Some steps that can be taken to reduce the risk of failures are also discussed.

Sammendrag

NTNU Test Satellitt er et student satellitt program på NTNU som er del av det norske student satellitt programmet drevet av nasjonalt senter for romrelatert opplæring. Målet med prosjektet er å sende en student bygget og designet Cubesat til verdensrommet. Denne oppgaven fortsetter arbeidet med å utvikle et Attitude Determination and Control System for NTNU Test Satellitt.

I denne oppgaven har simuleringer blitt kjørt for de tidligere valgte kontrollerne (detumbling og pointing) for å sjekke at de virker med hardwaren og softwaren designet til Attitude Determination and Control Systemet. En tredje kontroller har blitt utviklet for å øke vinkelhastigheten til satellitten, kalt en tumbling kontroller, simulering har også blitt utført for å teste denne. Simulering har også blitt gjort for å sjekke hvordan poining kontrolleren virker når det er målestøy på vinkelhastigheten. Simuleringen har vist at alle kontrollerne virker. Bevis på ustabilitet er også presentert for tumbling kontrolleren.

En ny prototype har blitt utviklet for Attitude Determination and Control System hardwaren med et fokus på redundans. Den overordnede software arkitekturen har også blitt designet. Mesteparten av softwaren har blitt testet på et evaluasjons kit for den valgte microcontrolleren. En median voterings algoritme har blitt implementert for å behandle redundansen i software. Til sist har en kort feil analyse blitt gjennomført, feiltrær har blitt brukt for å visse hvordan Attitude Determination and Control System kan slutte å fungere som designet. Noen alternativer for å redusere risikoen for feil har også blitt diskutert.

Acknowledgement

I would like to thank Henrik Rudi Haave and Antoine François Xavier Pignède who have also worked on the Attitude Determination and Control System for the NTNU Test Satellite for their support and cooperation this last year.

I would also like to thank Amund Gjersvik who works on the NTNU Test Satellite project for helping with design of the Attitude Determination and Control System hardware, and the project leader Roger Birkeland for his help.

Lastly I would like to thank my supervisor professor Jan Tommy Gravdahl for the help and guidance he has given during the semester, and professor Amund Skavhaug for the invaluable guidance on the software side.

Contents

List of Figures

List of Tables

Acronyms

I^2C Inter-Integrated Circuit. 54, 59

ADC Analog-to-Digital-Converter. 53

ADCS Attitude Determination and Control System. 1–6, 16, 17, 19, 29, 30, 47, 51–55, 57, 59–61, 63, 66–69, 71, 72, 75, 77–81, 83, 85

ANSAT Norwegian Student Satellite Program. 1

COTS Commercial Off The Shelf. 1, 16

CSP Cubesat Space Protocol. 53, 57, 59

ECEF Earth-Centered Earth-Fixed. 8

ECI Earth-Centered Inertial. 7, 8, 13, 33

EKF Extended Kalman Filter. 3

EPS EPS. 2, 55

EQUEST Extended Quaternion Estimator. 3

FIFO First In First Out. 52, 58

FMEA Failure Mode and Effects Analysis. 81

FreeRTOS Free Real Time Operating System. 57, 58, 64, 66, 83

IGRF International Geomagnetic Reference Field. 34

JTAG Joint Test Action Group. 54

MSS Marine Systems Simulator. 33, 34

NAROM Norwegian Center for Space Related Education. 1

NASA National Aeronautics and Space Administration. 16

NRP NUTS Reliable Protocol. 59, 63, 64, 66, 80, 85

NTNU Norwegian University of Science and Technology. 1

NUTS NTNU Test Satellite. 1, 3, 5, 8, 10, 11, 16, 17, 19, 21, 22, 33, 47, 53, 57, 59, 60, 77–79

OBC On Board Computer. 2, 53, 55, 57, 59, 60, 80

P-POD Poly-Picosatellite Orbital Deployer. 38

PD Proportional-Derivative. 23

PWM Pulse Width Modulation. 3, 20, 28, 29, 33

RTC Real-Time Clock. 53, 75, 78

SEB Single Event Burnout. 17

SEGR Single Event Gate Rupture. 17

SEL Single Event Latchup. 17

SEP Single Event Phenomena. 16, 72, 74

SET Single Event Transient. 17

SEU Single Event Upset. 17

SPI Serial Peripheral Interface. 52, 53

USART Universal Synchronous/Asynchronous Receiver/Transmitter. 54

Chapter 1

Introduction

1.1 NUTS

The Norwegian University of Science and Technology (NTNU) test satellite project, called NTNU Test Satellite (NUTS), was started in 2010. The goal of the project is to send a student built and developed double (2U) CubeSat [1] into space. NUTS is part of the Norwegian Student Satellite Program (ANSAT), which is run by the Norwegian Center for Space Related Education (NAROM). NUTS will be the fifth CubeSat built in Norway by students after N-CUBE 1, N-CUBE 2, HinCube and CubeSTAR. N-CUBE 1 and 2 where nationally built and much of the experience gained from these projects have been included in NUTS.

NUTS is a student driven cross disciplinary research and development project at NTNU, with students from fields such as electronics, communications, physics, cybernetics and mechanics. Many of the parts are being built at NTNU, the remaining are mostly Commercial Off The Shelf (COTS) components. Originally, NUTS was planned to be equipped with an infrared camera as payload, but due to cost considerations this idea was scrapped and the payload was changed to a regular camera. Unfortunately, due to a lack of students to work on it the camera will not be finished in time. As a consequence, the Attitude Determination and Control System (ADCS) that was originally developed with the purpose of pointing the camera at Earth will no longer serve this function. The system will however still be finished and used on the satellite since it can be considered a payload in itself and there is a lot of learning value for the students and any future CubeSat projects. The work done here and for the ADCS previously can be used directly in another CubeSat in the future.

The satellite consists of 5 modules connected to a backplane, an example of what this might look like is given in Figure 1.1.

- Mechanical System
- On Board Computer (OBC)
- Attitude Determination and Control System (ADCS)
- EPS (EPS)
- Communication System

In addition there is also a ground station segment to track and control the satellite.

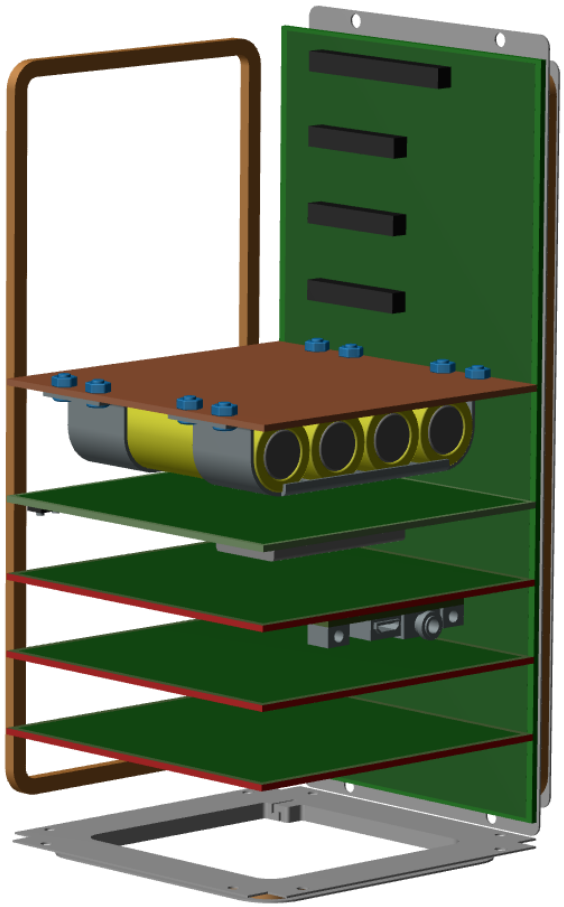

Figure 1.1: Concept for the inside of NUTS

1.2 Previous Work

The NUTS project was initiated in 2010 and a lot of work has been done on the project so far. Several different control strategies have been investigated and simulated in [2, 3, 4, 5]. Most of the theoretical work for the controllers is built on what was found in [6]. Attitude determination algorithms like the Extended Quaternion Estimator (EQUEST) and a Extended Kalman Filter (EKF) have been explored in [7, 8, 9]. The first prototype for the ADCS was developed in [10].

Two other master student have been working on the ADCS in parallel with the work presented here. Henrik Rudi Haave have been working on the sun sensors used for the ADCS and design of the new hardware in cooperation with myself and Amund Gjersvik. Unfortunately the thesis written by Haave will be handed in later, so any references to his work in this thesis will simply be done by using his name. Antoine François Xavier Pignède has developed an algorithm to find out where the satellite is in orbit, estimate a magnetic field vector, estimate a sun vector, and find the satellites orbital speed. In addition he has investigated robust spacecraft attitude stabilization [11].

1.3 Contributions of this Thesis

The main focus for the NUTS project in this stage has been to finish as many components as possible. This master thesis and the other theses written for the ADCS this semester are thus focused on finishing different parts of the system. Here the main contributions from the work described in this thesis are listed.

Pulse Width Modulation Evaluation

Several more simulations for the chosen controllers have been performed to see how they perform with the one cycle Pulse Width Modulation (PWM) actuator design in a discrete environment where the time period is relatively long.

Pointing controller robustness

Simulations have been performed for the pointing controller with noise added to the angular measurements. This have been done since noisy angular velocity measurements were a problem in [5].

Tumbling controller

A tumbling controller have been found and proven unstable, this will be used to increase the angular velocity of the satellite. Simulations have been performed to check the performance of the controller.

Attitude Determination and Control System hardware prototype

A second ADCS prototype board has been developed in cooperation with Henrik Ruudi Haave and Amund Gjersvik based on the first prototype. Changes and improvements have been made to make a flight ready version. The main difference between the first and second prototype is that more sensors have been added for redundancy.

Attitude Determination and Control System Software design

The overall software architecture for the ADCS has been designed. This includes definitions of which data structures are going to be used, how they are being shared between modules, and which subsystems goes in which task. A definition for which messages the ADCS can send and receive has also been made.

Sensor redundancy algorithm

Algorithms to handle sensor redundancy for gyroscope and magnetometer measurements have been investigated, and a median voter algorithm has been implemented.

Failure Analysis

A short failure analysis for the ADCS is also presented. Fault trees have been used to show how the ADCS might fail to perform what it have been designed to. A discussion on how to reduce the likelihood of errors are also included. Nothing has been implemented in software, but it is meant as a guide for any further development of the software.

A thorough effort has been made to clear up some of the problems with previous works that this thesis is built upon, and to summarize where the design of the ADCS stands. This is especially true for the control algorithms and the hardware, since no more work should be needed on these systems after this thesis.

The ADCS is a project that relies on several people. This has lead to delays that have affected this thesis, like the hardware not being delivered, and some software subsystems being finished late or not at all. Some tasks that might have seemed natural to include in this thesis

have therefore not been performed, tasks such as timing of the software subsystems and developing new drivers for the new hardware. That being said, everything presented here is of use to the NUTS project and helps bring the ADCS closer to completion.

1.4 Thesis Outline

Chapter 2 focuses on the basic mathematical principles that are being used for modeling and simulation of the satellite. This includes an explanation of the rotation matrix, Euler angles and quaternions.

Chapter 3 introduces the satellites dynamics, explains how the magnetorquers work, describes the forces working on the satellite in space and some important theory for a failure analysis is introduced.

Chapter 4 presents the different controllers that have been used in this project. A pointing controller, a detumbling controller and a tumbling controller. It also shortly describes pulse width modulation and presents instability evidence for the tumbling controller.

Chapter 5 presents the simulation results for all the controllers. An analysis of the results for each individual controller is also included. This includes the simulations of the pointing controller where noise has been added to the angular velocity measurements.

Chapter 6 presents the changes done on the hardware for this second ADCS prototype. The important changes and additions are listed and shown.

Chapter 7 discusses some software concepts, which are then used to define a software architecture for the ADCS. The different subsystems are listed, how data should be stored and shared are defined and two architectures are presented.

Chapter 8 investigates some options for processing the sensor data from the gyroscopes and magnetometers to find the best value. The demands of such an algorithm is defined, and some different algorithms are presented. A software implementation for a median voter al-

gorithm is also included.

Chapter 9 investigates how the ADCS might fail and how failures can be handled. Fault trees are being used to show how the ADCS may fail in doing what it was designed to.

Chapter 10 discusses and summarizes the results that have been presented trough this thesis.

Chapter 11 presents the conclusion of this thesis.

Chapter 12 focuses on what has to be done in the future to finish the ADCS.

Chapter 2

Background Theory

Most of this chapter is taken from [5] (the authors 5th year project), with modifications to suit this thesis. This chapter will cover reference frames, the rotation matrix, Euler angles and quaternions.

2.1 Reference Frames

A reference frame is a coordinate system that is being used to represent the position and orientation of an object. For satellite navigation it is clearly necessary to specify where the satellite is, and how it is oriented (what its attitude is), but this has to be done with respect to some other point in space, reference frames allows us to do this. Which frame one choose to use depends on what one is trying to describe.

2.1.1 Earth-Centered Inertial Frame

The Earth-Centered Inertial (ECI) frame is a coordinate frame that has its origin at the center of the Earth. Its axis are defined by the vectors $\{i\} = (x_i, y_i, z_i)$. The z_i axis is defined as a vector directed from the center of the Earth towards the celestial north pole. The x_i axis points in the vernal equinox direction. This is a vector pointing from the center of Earth to the center of the Sun on the vernal equinox. The vernal equinox is the time of year when the Sun crosses the Earths equatorial plane going from south to north, the sun is then directly over the equator. I.e the center of the Sun lies in the same plane as the Earths equator. The y_i axis is defined by the right hand rule to complete the coordinate system.

2.1.2 Earth-Centered Earth-Fixed Frame

The Earth-Centered Earth-Fixed (ECEF) frame is like the ECI frame except that it is rotational. Its axis are defined by the vectors $\{e\} = (x_e, y_e, z_e)$. The z_e axis is defined as a vector directed from the center of the Earth towards the celestial north pole. The x_i axis points in the direction of 0° longitude and 0° latitude, thus the frame rotates with Earth around the z_e axis with a angular speed of $\omega_e = 7.2921 \times 10^{-5} \frac{rad}{sec}$. Again the y_e axis is defined by the right hand rule to complete the coordinate system.

2.1.3 Orbit Frame

The orbit frame is a moving coordinate frame that has its origin at the center of the satellite. Its axis is defined by the vectors $\{o\} = (x_o, y_o, z_o)$. The z_o axis is defined as a vector pointing in the nadir direction (towards the center of the Earth). The x_o axis is defined so that it points in the orbit normal direction, e.g along the satellites linear velocity vector. Again the y_o axis is defined by the right hand rule to complete the coordinate system. This definition of the orbit frame is only valid for a circular orbit.

2.1.4 Body-Fixed Frame

The body-fixed frame is also a moving coordinate frame with its origin at the center of the satellite. Its axis is defined by the vectors $\{b\} = (x_b, y_b, z_b)$. Unlike the orbit frame the body-fixed frame is fixed to the satellite, as the name implies, it moves and rotates with the satellite. For NUTS it have been define so that z_b is the axis of minimum inertia, y_b is the axis of maximum inertia and the x_b axis is defined by the right hand rule to complete the coordinate system. x_b is the roll axis, y_b is the pitch axis and z_b is the yaw axis. The reasoning behind this choice of axis follows from the stability analysis done in [6]. This frame is also referred to as the body frame.

2.2 Attitude Representation

The attitude of a satellite (or any craft) describes how the craft is oriented relative to some reference frame. In this thesis, the rotation matrix, Euler angles and Quaternions are being used to represent the satellites attitude and they will be defined here.

2.2.1 The Rotation Matrix

The rotation matrix is used to rotate or transform a vector from one reference frame to another, or to rotate a vector within a reference frame. A rotation of the vector v from frame a to b is denoted as $v^b = \mathbf{R}_a^b v^a$. The rotation matrix is an element in $SO(3)$ which is formally defined as:

$$SO(3) := \left\{ \mathbf{R} \mid \mathbf{R} \in \mathbb{R}^{3\times 3}, \quad \mathbf{R}^T\mathbf{R} = \mathbf{I}, \quad \det(\mathbf{R}) = 1 \right\} \tag{2.1}$$

A useful way to look at the rotation matrix is to describe it as a rotation β about a unit vector $\boldsymbol{\lambda}$ as follows:

$$\mathbf{R}_{\boldsymbol{\lambda},\beta} = \mathbf{I}_{3\times 3} + \sin(\beta)\mathbf{S}(\boldsymbol{\lambda}) + \left[1 - \cos(\beta)\right]\mathbf{S}^2(\boldsymbol{\lambda}), \quad \boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} \tag{2.2}$$

Where **S** is a skew-symmetric matrix defined as:

$$\mathbf{S}(\boldsymbol{\lambda}) = -\mathbf{S}(\boldsymbol{\lambda})^T = \begin{bmatrix} 0 & -\lambda_3 & \lambda_2 \\ \lambda_3 & 0 & -\lambda_1 \\ -\lambda_2 & \lambda_1 & 0 \end{bmatrix}, \quad \boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix} \tag{2.3}$$

The skew-symmetric matrix, **S**, is also called a cross product operator:

$$\mathbf{S}(\boldsymbol{\lambda})\mathbf{a} = \boldsymbol{\lambda} \times \mathbf{a}, \quad \boldsymbol{\lambda} = \begin{bmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{bmatrix}, \quad \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ |a_3 \end{bmatrix} \tag{2.4}$$

For more on the rotation matrix see Chapter 6 in [12].

The rotation matrix from the body frame b, to the orbit frame o, can be written as:

$$\mathbf{R}_o^b = \begin{bmatrix} x_b \cdot x_o & x_b \cdot y_o & x_b \cdot z_o \\ y_b \cdot x_o & y_b \cdot y_o & y_b \cdot z_o \\ z_b \cdot x_o & z_b \cdot y_o & z_b \cdot z_o \end{bmatrix} = \begin{bmatrix} c_{11} & c_{12} & c_{13} \\ c_{21} & c_{22} & c_{23} \\ c_{31} & c_{32} & c_{33} \end{bmatrix} = \begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \mathbf{c}_3 \end{bmatrix} \tag{2.5}$$

The elements of the first matrix are the x, y and z components defining the satellites attitude in the body and orbit frame, as defined in Chapter 2.1. When talking about the satellites attitude we are referring to how the body frame is oriented relative to the orbit frame. This

rotation matrix can thus be used to describe the satellites attitude. The elements in the second matrix are called direction cosines, that is they are the cosines of the angles between the two vectors from the first matrix. Direction cosines can also be used to represent attitude, but they have not been used in the NUTS project, except to prove stability or instability.

2.2.2 Euler Angles and Quaternions

The most common way to describe the attitude of any system is by using the Euler angles, roll (ϕ), pitch (θ) and yaw (ψ). As mentioned earlier, for this satellite roll describes rotation around the x_b axis, pitch describes rotation around the y_b axis and yaw describes rotation around the z_b axis. The Euler angle vector is:

$$\mathbf{\Theta} = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \tag{2.6}$$

The already mentioned rotation matrix can, with this notation, be rewritten into a set of principal rotations defined as:

$$\mathbf{R}_{x,\phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{2.7}$$

$$\mathbf{R}_{y,\theta} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{2.8}$$

$$\mathbf{R}_{z,\psi} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

$\mathbf{R}_{x,\phi}$ is a rotation of ϕ around the x-axis, $\mathbf{R}_{y,\theta}$ is a rotation of θ around the y-axis and $\mathbf{R}_{z,\psi}$ is a rotation of ψ around the z-axis. The full rotation matrix can be written as:

$$\mathbf{R}_o^b = \mathbf{R}_{x,\phi}\mathbf{R}_{y,\theta}\mathbf{R}_{z,\psi} \tag{2.10}$$

This attitude representation is easy to understand, but it can lead to singularities in the equations of motion. For example at a pitch angle of $\theta = \pm 90°$ a singularity will appear in the equations of motion, which is a big problem. For our satellite system, that can rotate more than 90° in all directions, this is not a suitable way to describe the attitude in a controller.

The alternative used for this project are quaternions, they are harder to interpret directly but will not lead to any singularities and are thus much better to use for a satellite. The quaternions are a number system that extends the complex numbers, a quaternion, $\mathbf{q}$, consists of one real part η and three imaginary parts $\boldsymbol{\epsilon} = \begin{bmatrix}\epsilon_1, \epsilon_2, \epsilon_3\end{bmatrix}^T$. The set Q of unit quaternions can be formally defined as:

$$Q := \left\{ \mathbf{q} \mid \mathbf{q}^T\mathbf{q} = 1, \quad \mathbf{q} = \begin{bmatrix}\eta, \boldsymbol{\epsilon}^T\end{bmatrix}^T, \quad \boldsymbol{\epsilon} \in \mathbb{R}^3 \text{ and } \eta \in \mathbb{R} \right\} \tag{2.11}$$

The unit quaternion is defined as:

$$\eta := \cos\left(\frac{\beta}{2}\right) \tag{2.12}$$

$$\boldsymbol{\epsilon} = \begin{bmatrix}\epsilon_1, \epsilon_2, \epsilon_3\end{bmatrix}^T = \boldsymbol{\lambda} \sin\left(\frac{\beta}{2}\right) \tag{2.13}$$

where $\boldsymbol{\lambda} = \begin{bmatrix}\lambda_1, \lambda_2, \lambda_3\end{bmatrix}^T$ is a unit vector that satisfies:

$$\boldsymbol{\lambda} = \pm \frac{\boldsymbol{\epsilon}}{\sqrt{\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}}} \quad \text{if} \quad \sqrt{\boldsymbol{\epsilon}^T\boldsymbol{\epsilon}} \neq 0 \tag{2.14}$$

This is the same $\boldsymbol{\lambda}$ and β as in Equation 2.2. The definition given above is that of a unit quaternion representing a rotation matrix through its Euler parameters. Throughout the NUTS project the unit quaternion as defined above has often been referred to only as a quaternion. For more on Euler angles and quaternions see Chapter 6 in [12].

Chapter 3

Satellite Theory

Most of this chapter is taken from [5], with modifications to suit this thesis. In this chapter the satellite dynamics and kinematics is presented. The theory behind the magnetic actuators and how they work is explained. Furthermore, the environmental disturbances that can affect the satellite is presented. Lastly, some important theory for a failure analysis is presented.

3.1 Satellite Dynamics and Kinematics

Treating the satellite as a rigid body its equation of motion for rotational movement can be expressed as [6]:

$$\boldsymbol{\tau}^b = \mathbf{I}\dot{\boldsymbol{\omega}}_{ib}^b + \boldsymbol{\omega}_{ib}^b \times (\mathbf{I}\boldsymbol{\omega}_{ib}^b) \tag{3.1}$$

Where $\boldsymbol{\omega}_{ib}^b$ is the angular velocity of the satellite in the body frame with respect to the ECI frame given in the body frame. $\mathbf{I}$ is the satellites moment of inertia matrix around its mass center and $\boldsymbol{\tau}^b$ is the moment affecting the satellite, given in the body frame. The angular velocity of the satellite, $\boldsymbol{\omega}_{ib}^b$, can be expressed as:

$$\boldsymbol{\omega}_{ib}^b = \boldsymbol{\omega}_{ob}^b + \boldsymbol{\omega}_{io}^b = \boldsymbol{\omega}_{ob}^b + \mathbf{R}_o^b \boldsymbol{\omega}_{io}^o \tag{3.2}$$

Where $\boldsymbol{\omega}_{io}^o = [0,\ -\omega_o,\ 0]$ and $\omega_o = \sqrt{\frac{GM_E}{R_o^3}}$ is the satellites orbital speed. G is the gravitational constant, M_E is the mass of the earth and $R_o = R_s + R_E$, where R_E is the earths radius and R_s is the satellites altitude. Here $\boldsymbol{\omega}_{ob}^b$ is the angular velocity of the satellite in the body frame with respect to the orbit frame, given in the body frame.

The kinematic differential equation for quaternions are given as [13]:

$$\dot{\eta} = -\frac{1}{2}\boldsymbol{\epsilon}^T \boldsymbol{\omega}_{ob}^b \tag{3.3}$$

$$\dot{\boldsymbol{\epsilon}} = \frac{1}{2}(\eta \mathbf{I} + \mathbf{S}(\boldsymbol{\epsilon}))\boldsymbol{\omega}_{ob}^b \tag{3.4}$$

3.2 Magnetorquers

The satellite uses magnetic actuators, called magnetorquers. These magnetorquers are coils that have been wound up a number of times. The satellite will have three magnetorquers mounted in alignment with the defined body frame, one on the z-axis, one on the y-axis, and one on the x-axis. Running current trough the coils creates a magnetic moment **m** perpendicular to the coil that reacts with the magnetic field **B** and creates torque according to:

$$\boldsymbol{\tau}^b = \mathbf{m}^b \times \mathbf{B}^b \tag{3.5}$$

Since all the measurements and calculations are either being done in or transformed to the body frame the body superscript is used. To find the direction of the magnetic moment one can look at the current and use the right-hand rule. The magnetic field created by the coils will attempt to align itself with the local geomagnetic field, this will in turn create torque, making the satellite rotate. The magnetic moment is given by:

$$\mathbf{m}^b = \begin{bmatrix} N_x i_x A_x \\ N_y i_y A_y \\ N_z i_z A_z \end{bmatrix} \tag{3.6}$$

Where N is the number of windings in the coil, A is the area the coil covers and i is electric current. A problem that comes with using magnetorquers is that it gives us an underactuated system if one of the axis are aligned with the local magnetic field. If that happens and the goal is to turn the satellite around that axis the coils will provide no torque and the system will be underactuated. This leads to the system being uncontrollable whenever one of the axis are parallel to the local magnetic field.

The satellites magnetorquers developed in [10] have the specifications given in Table 3.1. The table is included again here since some of these values are needed in the simulations. Notice that the X- and Y-coils are identical, while the Z-coil is different, this is so that the actuators creates the same magnetic moment when the same power is applied to all three. Since $A_z < (A_x = A_y)$ the following must be true, $i_z > (i_x = i_y)$ and/or $N_z > (N_x = N_y)$, to produce the same magnetic moment, as can be seen from Equation 3.6.

Table 3.1: Magnetorquer specifications, taken from [10]

Parameter	**X and Y-coil**	**Z-coil**
Windings	155	238
R theoretical	26.7 Ω	19.2 Ω
R measured	27.2 Ω	18.5 Ω
f	590 Hz	580 Hz
L	8.53 mH	9.13 mH
τ	313 μs	494 μs

3.3 Environmental Disturbances

Even though the satellite is going to operate in a low Earth orbit, there are several environmental factors that are going to affect it and create disturbance torques. The most prominent disturbances are:

- Gravity
- Aerodynamic torque
- Solar radiation torque
- Internal magnetic dipole moments and other electrical noise

The most important disturbance to model is gravity, since the controller is based on the satellite working with the torque generated by the gravity of Earth. Simulations have been done with all these disturbances in [2].The simulations that have been performed here have been done with the gravity gradient torque only.

For an unsymmetrical satellite the Earths gravity will pull differently on different parts of

the satellite, creating torque according to [14]:

$$\boldsymbol{\tau}_g^b = 3\omega_o^2 \mathbf{c}_3 \times (\mathbf{I}\mathbf{c}_3) \tag{3.7}$$

Where ω_o is the satellites orbital speed, and $\mathbf{c}_3$ is a vector of direction cosines from the rotation matrix given in equation 2.5. This is the gravity gradient torque.

3.4 Error Sources

This section is based on the work done for NUTS in [15]. One of the main errors sources for satellites is the hard radiation operating environment [16]. Chapter 9 deals with how the ADCS might fail, and how it can be avoided. Here the necessary theory for such a analysis is introduced. Some important terminology is given below, this is taken from [17].

Failure is an event that occurs when the delivered service deviates from the specified service, failures are caused by errors.

Error is the manifestation of a fault within a program or data structure, errors can occur some distance from the fault sites.

Fault is an incorrect state of hardware or software resulting from failures of components, physical interference from the environment, operator error, or incorrect design.

Permanent describes a failure or fault that is continuous and stable, in hardware permanent failures reflect an irreversible physical change.

Intermittent describes a fault that is only occasionally present due to unstable hardware or varying hardware or software states.

Transient described a fault resulting from temporary environmental conditions.

Radiation errors are divided into two classes, Single Event Phenomena (SEP) and total radiation dosage effects. Table 3.2 shows the different kinds of SEP that can occur. While a SEP can occur at any time, total radiation dosage errors will occur after the components have received a certain amount of radiation. NUTS, and most CubeSats, use only self made or COTS components, these are especially sensitive to the different kinds of space radiation. The National Aeronautics and Space Administration (NASA) estimates that COTS components will

Table 3.2: Single Event Phenomena, taken from [15]

Name	Effect
Single Event Transient (SET)	Soft intermittent fault Propagating through circuit
Single Event Upset (SEU)	Soft transient fault State change on latch or memory Often refereed to as a bit-flip
Single Event Latchup (SEL)	Apparent short circuit Can be mitigated with power cycling Can cause destructive thermal runaway
Single Event Gate Rupture (SEGR)	Permanent Failure
Single Event Burnout (SEB)	Permanent Failure

suffer from a SEU error rate of 10^{-5} errors/bit-day and tolerate a total radiation dose of about 2 to 10 krad (100rad = 1gray(Gy) = $1\frac{J}{kg}$) [16].

SEGR and SEB failures are hard to counteract and have not been dealt with for NUTS. SEL failures can mostly be cleared by power cycling the ADCS, a function that is already built into the backplane of NUTS [15]. SET and SEU errors can to a certain extent be counteracted in software. Total radiation dosage effects can postponed by shielding the components. For a more detailed description of space radiation effects and how a CubeSat can be built to detect and correct such errors see [15].

A common way to reduce the likelihood of critical systems failing is to include redundancy in those systems. Redundancy is defined as the inclusion of extra components so that the system can continue to function as specified even if some components stop working as they should. It is important to note that having redundancy can also greatly increase the complexity and maintainability of a system, since having more components increases the amount of errors being introduced [18].

Chapter 4

Control

Parts of this chapter have been taken from [5]. The target of the ADCS is to point the satellites positive z-side towards Earth, this is the case when the body frame is aligned with the orbit frame. This can be expressed in an Euler angle as:

$$\boldsymbol{\Theta} = [0\ 0\ 0]^T \tag{4.1}$$

Or in a quaternion as:

$$\mathbf{q} = [1\ 0\ 0\ 0]^T \tag{4.2}$$

Usually the term nadir pointing is used for this because of the definition of the orbit frame. The maximum error allowed on the pointing accuracy and drift rate while still considering the ADCS to be pointing in the nadir direction is given in Table 4.1.

Table 4.1: Attitude Control System Requirements

	Roll	**Pitch**	**Yaw**
Pointing Accuracy [deg]	± 25	± 25	–
Angulvar Velocity [rad/sec]	$\pm 1 \cdot 10^{-3}$	$\pm 1 \cdot 10^{-3}$	$\pm 1 \cdot 10^{-3}$

Several simulations have already been performed for the pointing controller and the detumbling controller, [2, 19, 3], but no one has investigated how the controllers will work with the actuator design on NUTS. One of the main goals of this thesis is to investigate how the controllers will behave in a discrete environment with the current actuator design. The actu-

ators (the magnetorquers) are controlled by using a one cycle PWM. The behavior of the controllers will therefore be investigated by running simulations with a one cycle PWM control output. Here the different controllers will be presented again, as well as a short description of PWM.

4.1 B-Dot Estimator

Because of some issues with how the B-Dot estimator was presented in [2] it is repeated here. For the detumbling controller $\dot{\mathbf{B}}^b$ is needed, but this cannot be measured directly. An estimator to calculate $\dot{\mathbf{B}}^b$ from $\mathbf{B}^b$ has been developed. The analysis for the estimator is mostly given in the Laplace domain and is presented below.

$\dot{\mathbf{B}}$ can be found from $\mathbf{B}^b$ in the following way:

$$H(s) = \frac{\dot{\mathbf{B}}^b}{\mathbf{B}^b} = s \tag{4.3}$$

The measurements of $\mathbf{B}^b$ might be noisy, so a low-pass filter is also desired:

$$G(s) = K\frac{\omega_c}{s+\omega_c} \tag{4.4}$$

The estimator is then given by:

$$J(s) = H(s)G(S) = \frac{\hat{\dot{B}}}{B} = K\frac{\omega_c s}{s+\omega_c} \tag{4.5}$$

This is a first order state variable filter. A discrete version of this estimator is needed, to find it the estimator is first transformed to the z-domain:

$$J(z) = \frac{\hat{\dot{B}}}{B} = K\omega_c\frac{z-1}{z-e^{-\omega_c T_s}} = \frac{b(1-z^{-1})}{1-az^{-1}} \tag{4.6}$$

This can be rewritten to:

$$\hat{\dot{B}}(1-az^{-1}) = b(1-z^{-1})B \tag{4.7}$$

The discrete estimator can then be written as a recurrent filter:

$$\hat{\dot{\mathbf{B}}}_k = a\hat{\dot{\mathbf{B}}}_{k-1} + b(\mathbf{B}_k - \mathbf{B}_{k-1}) \tag{4.8}$$

Here ω_c is the cut-off frequency, K is the filter passband gain and T_s is the sampling time. A good value for the cut-off frequency has been found to be $\omega_c = 0.7$ [2, 20]. The sampling time, T_s, has varied with the discrete period used in the simulations. The filter coefficients are given by:

$$a = e^{-\omega_c T_s} \tag{4.9}$$

$$b = K\omega_c \tag{4.10}$$

4.2 Detumbling Controller

The detumbling controller chosen for NUTS is the B-dot controller first proposed in [21]. It has since become a popular controller in the CubeSat community, where it has taken slightly different forms, [22, 6]. The B-dot control law used by NUTS is [6]:

$$\mathbf{m}^b = -k\dot{\mathbf{B}}^b \quad , \quad k > 0 \tag{4.11}$$

$\mathbf{m}^b$ is the magnetic dipole moment, k is the positive control gain and $\dot{\mathbf{B}}^b$ is the time derivative of the local magnetic field.

Since we cannot measure $\dot{\mathbf{B}}^b$ directly the estimate is used:

$$\mathbf{m}^b = -k\hat{\dot{\mathbf{B}}}^b \quad , \quad k > 0 \tag{4.12}$$

[6] suggests choosing the gain, k, as:

$$k = \frac{d}{|\mathbf{B}^b|^2}, \quad d > 0 \tag{4.13}$$

With this gain the B-dot detumbling control law becomes:

$$\mathbf{m}^b = -\frac{d}{|\mathbf{B}^b|^2}\hat{\dot{\mathbf{B}}}^b, \quad d > 0 \tag{4.14}$$

A stability analysis for this controller has been presented in [2] and is therefore not repeated here. The stability analysis finds that energy is dissipated so that the angular velocity will decrease. It is found that the satellites inertia matrix needs to satisfy the following equation:

$$I_y > I_x > I_z \tag{4.15}$$

4.3 Pointing Controller

The pointing controller chosen for NUTS is based on the work in [6]. The derivation of this control law from an ideal controller is shown here. A controller on the following form is desired:

$$\boldsymbol{\tau}_d^b = -p\boldsymbol{\epsilon} - d\boldsymbol{\omega}_{ob}^b \quad , \qquad p, d > 0 \tag{4.16}$$

p and d are constant positive gains, $\boldsymbol{\epsilon}$ equals the three last elements of the quaternion vector as defined in Chapter 2 and $\boldsymbol{\omega}_{ob}^b$ is the angular velocity of the satellites body frame with respect to the orbit frame given in the body frame.

Unfortunately, this controller is completely hypothetical since the magnetorquers cannot be used to set up an arbitrary magnetic moment. The torque is given by $\boldsymbol{\tau}^b = \mathbf{m}^b \times \mathbf{B}^b$, this will be in the plane that $\mathbf{B}^b$ is normal to. A solution to this is to project $\boldsymbol{\tau}_d^b$ so that it is normal to $\mathbf{B}^b$. Using the controller in Equation 4.16 the actual torque would still be in the plane normal to the magnetic field, but for the controller to be as efficient as possible it should be projected to the right plane.

The direction of the magnetic moment $\mathbf{m}^b$ is given by the cross product between $\mathbf{B}^b$ and $\boldsymbol{\tau}_d^b$, multiplied by some f to get the right length:

$$\mathbf{m}^b = f\mathbf{B}^b \times \boldsymbol{\tau}_d^b \tag{4.17}$$

The torque is then given by:

$$\boldsymbol{\tau}^b = f(\mathbf{B}^b \times \boldsymbol{\tau}_d^b) \times \mathbf{B}^b \tag{4.18}$$

Defining the angle between $\mathbf{B}^b$ and $\boldsymbol{\tau}_d^b$ as α the length of $\boldsymbol{\tau}^b$ can be defined as:

$$|\boldsymbol{\tau}^b| = |\mathbf{m}^b||\mathbf{B}^b| = f|\mathbf{B}^b|\,|\boldsymbol{\tau}_d^b|\,|\mathbf{B}^b|\sin\alpha \tag{4.19}$$

This length can also be defined as:

$$|\boldsymbol{\tau}^b| = |\boldsymbol{\tau}_d^b|\sin\alpha \tag{4.20}$$

From this it is clear that f is:

$$f = \frac{1}{|\mathbf{B}^b|^2} \tag{4.21}$$

This is also the reason for choosing the gain, k, in Equation 4.13 in this way. The resulting pointing controller becomes:

$$\mathbf{m}^b = -\frac{1}{|\mathbf{B}^b|^2}\left(p(\mathbf{B}^b \times \boldsymbol{\epsilon}) + d(\mathbf{B}^b \times \boldsymbol{\omega}_{ob}^b)\right), \qquad p,d > 0 \tag{4.22}$$

This is a nonlinear Proportional-Derivative (PD) controller. Linearization has shown that the controller is locally stable [6], but unfortunately no clear stability evidence has been found. It is found that the satellites inertia matrix should satisfy the following equation:

$$I_y > I_x > I_z \tag{4.23}$$

Efforts have been made to find a better stability evidence for this controller, unfortunately there have been no results.

4.4 Tumbling Controller

A controller to increase the angular velocity of the satellite is presented here. This might not seem useful at first glance, but the idea is to check how components like the radio performs when the satellite is spinning. A tumbling controller could simply be a feedback from the angular velocity:

$$\boldsymbol{\tau}_d^b = g\boldsymbol{\omega}_{ob}^b, \quad h > 0 \tag{4.24}$$

Here $\boldsymbol{\omega}_{ob}^b$ is the angular velocity of the satellites body frame with respect to the orbit frame given in the body frame. Just like for the pointing controller this control expression should

be projected to the plane normal to $\mathbf{B}^b$, the procedure is exactly the same as for the pointing controller. The tumbling controller can then be expressed as:

$$\mathbf{m}^b = h\frac{1}{|\mathbf{B}^b|^2}(\mathbf{B}^b \times \boldsymbol{\omega}_{ob}^b) \quad , \quad h > 0 \tag{4.25}$$

A short analysis to prove that this controller will actually increase the angular velocity of the satellite is included next.

4.4.1 Instability

This instability analysis is based on Lyapunov Theory. Here a Lyapunov function and its derivative will be found and used to prove that the controller in Equation 4.25 is unstable. The Lyapunov analysis is based on what was done in [6] and [14].

The potential energy for a rigid body in a circular orbit is [14]:

$$U = -\frac{GM_E m}{R_o} - \frac{1}{2}\omega_o^2(I_x + I_y + I_z) + \frac{3}{2}\omega_o^2 \mathbf{c}_3^T \mathbf{I}\mathbf{c}_3 \tag{4.26}$$

Similarly, the kinetic energy is [14]:

$$T = \frac{1}{2}m\omega_o^2 R_o^2 + \frac{1}{2}(\boldsymbol{\omega}_{ib}^b)^T \mathbf{I}\boldsymbol{\omega}_{ib}^b \tag{4.27}$$

The first term is the translational energy, and the second term is the rotational energy.

The kinetic energy, T, can be split into three parts by inserting a rewritten form of Equation 3.2:

$$\boldsymbol{\omega}_{ib}^b = \boldsymbol{\omega}_{ob}^b - \omega_o \mathbf{c}_2 \tag{4.28}$$

The kinetic energy is:

$$T = \frac{1}{2}m\omega_o^2 R_o^2 + \frac{1}{2}\omega_o^2 \mathbf{c}_2^T \mathbf{I}\mathbf{c}_2 - \omega_o \mathbf{c}_2^T \mathbf{I}\boldsymbol{\omega}_{ob}^b + \frac{1}{2}(\boldsymbol{\omega}_{ob}^b)^T \mathbf{I}\boldsymbol{\omega}_{ob}^b \tag{4.29}$$

This leads to:

$$T = T_0 + T_1 + T_2 \tag{4.30}$$

$$T_0 = \frac{1}{2} m\omega_o^2 R_o^2 + \frac{1}{2}\omega_o^2 \mathbf{c}_2^T \mathbf{I}\mathbf{c}_2 \tag{4.31}$$

$$T_1 = -\omega_o \mathbf{c}_2^T \mathbf{I}\boldsymbol{\omega}_{ob}^b \tag{4.32}$$

$$T_2 = \frac{1}{2}(\boldsymbol{\omega}_{ob}^b)^T \mathbf{I}\boldsymbol{\omega}_{ob}^b \tag{4.33}$$

There exists an energy function on the form [14]:

$$H = T_2 + (U - T_0) \tag{4.34}$$

$$H = \frac{1}{2}(\boldsymbol{\omega}_{ob}^b)^T \mathbf{I}\boldsymbol{\omega}_{ob}^b - \frac{GM_E m}{R_o} - \frac{1}{2}\omega_o^2(I_x + I_y + I_z) + \frac{3}{2}\omega_o^2 \mathbf{c}_3^T \mathbf{I}\mathbf{c}_3 - \frac{1}{2} m\omega_o^2 R_o^2 + \frac{1}{2}\omega_o^2 \mathbf{c}_2^T \mathbf{I}\mathbf{c}_2 \tag{4.35}$$

From this energy function a Lyapunov function can be derived [14]:

$$V = H - H_0 \tag{4.36}$$

H_0 is the value for H in the equilibrium of interest, this is $\boldsymbol{\omega}_{ob}^b = \mathbf{0}$ and $R_o^b = \mathbf{I}$. This is the attitude where the satellites body frame is aligned with its orbit frame and there are zero angular velocity.

$$H_0 = \frac{1}{2}\omega_o^2(3I_z - I_y) - \frac{1}{2} m\omega_o^2 R_o^2 - \frac{GMm}{R_o} - \frac{1}{2}\omega_o^2(I_x + I_y + I_z) \tag{4.37}$$

Equation 4.36 becomes:

$$\begin{aligned} V &= H - H_0 \\ &= \frac{1}{2}(\boldsymbol{\omega}_{ob}^b)^T \mathbf{I}\boldsymbol{\omega}_{ob}^b + \frac{3}{2}\omega_o^2(I_x c_{13}^2 + I_y c_{23}^2 + I_z(c_{33}^2 - 1)) - \frac{1}{2}\omega_o^2(I_x c_{12}^2 + I_y(c_{22}^2 - 1) + I_z c_{32}^2) \end{aligned} \tag{4.38}$$

Since one of the properties of the rotation matrix, $\mathbf{R}_o^b$, is that it is orthogonal, the following relationships are true: $c_{12}^2 + c_{22}^2 + c_{32}^2 = 1$ and $c_{13}^2 + c_{23}^2 + c_{33}^2 = 1$. Using these relationships to

simplify Equation 4.38 a Lyapunov function can be found to be:

$$\begin{aligned} V(\mathbf{x}) = &\frac{1}{2}(\boldsymbol{\omega}_{ob}^{b})^{T}\mathbf{I}\boldsymbol{\omega}_{ob}^{b} \\ &+\frac{3}{2}\omega_o^2(I_x - I_z)c_{13}^2 + (I_y - I_z)x_{23}^2 \\ &+\frac{1}{2}\omega_o^2(I_y - I_x)c_{12}^2 + (I_y - I_z)x_{32}^2 \end{aligned}$$

Define the state vector to be:

$$\mathbf{x} = \begin{bmatrix} (\boldsymbol{\omega}_{ob}^{b})^{T} & c_{13} & c_{23} & c_{12} & c_{32} \end{bmatrix}^{T} \tag{4.39}$$

This Lyapunov function satisfies $V(\mathbf{0}) = 0$ and $V(\mathbf{x}) > 0$ for $x \neq 0$ if:

$$I_y > I_x > I_z \tag{4.40}$$

This is also a requirement for the detumbling controller to be stable [2] and for the pointing controller [6].

To prove instability the time derivative of this Lyapunov function is needed:

$$\dot{V} = (\boldsymbol{\omega}_{ob}^{b})^{T}\mathbf{I}\dot{\boldsymbol{\omega}}_{ob}^{b} + 3\omega_o^2\mathbf{c}_3^{T}I\dot{\mathbf{c}}_3 - \omega_o^2\mathbf{c}_2^{T}I\dot{\mathbf{c}}_2 \tag{4.41}$$

The satellite dynamics from Equation 3.1 can be rewritten by also considering the gravity gradient torque from Equation 3.7:

$$\boldsymbol{\tau}^{b} = \mathbf{I}\dot{\boldsymbol{\omega}}_{ib}^{b} + \boldsymbol{\omega}_{ib}^{b} \times (\mathbf{I}\boldsymbol{\omega}_{ib}^{b}) - 3\omega_o^2\mathbf{c}_3 \times (\mathbf{I}\mathbf{c}_3) \tag{4.42}$$

Using Equation 4.28, Equation 4.42 can be written as:

$$\boldsymbol{\tau}^{b} = \mathbf{I}\dot{\boldsymbol{\omega}}_{ob}^{b} - \omega_o\mathbf{I}\dot{\mathbf{c}}_2 + \boldsymbol{\omega}_{ob}^{b} \times (\mathbf{I}\boldsymbol{\omega}_{ob}^{b}) - \omega_o\boldsymbol{\omega}_{ob}^{b} \times (\mathbf{I}\mathbf{c}_2) - \omega_o\mathbf{c}_2 \times (\mathbf{I}\boldsymbol{\omega}_{ob}^{b}) + \omega_o^2\mathbf{c}_2 \times \mathbf{I}\mathbf{c}_2 - 3\omega_o^2\mathbf{c}_3 \times \mathbf{I}\mathbf{c}_3 \tag{4.43}$$

Inserting this into Equation 4.41 the Lyapunov derivative can be rewritten to:

$$\dot{V} = (\boldsymbol{\omega}_{ob}^b)^T \left(\omega_o \mathbf{I}\dot{\mathbf{c}}_2 - \boldsymbol{\omega}_{ob}^b \times (\mathbf{I}\boldsymbol{\omega}_{ob}^b) + \omega_o \boldsymbol{\omega}_{ob}^b \times (\mathbf{I}\mathbf{c}_2) + \omega_o \mathbf{c}_2 \times (\mathbf{I}\boldsymbol{\omega}_{ob}^b) - \omega_o^2 \mathbf{c}_2 \times \mathbf{I}\mathbf{c}_2 + 3\omega_o^2 \mathbf{c}_3 \times \mathbf{I}\mathbf{c}_3 + \boldsymbol{\tau}^b \right) + 3\omega_o^2 \mathbf{c}_3^T I \dot{\mathbf{c}}_3 - \omega_o^2 \mathbf{c}_2^T I \dot{\mathbf{c}}_2 \quad (4.44)$$

A cross product of a vector with itself is the zero vector: $(\boldsymbol{\omega}_{ob}^b)^T \mathbf{S}(\boldsymbol{\omega}_{ob}^b) = \mathbf{0}$. The time derivative of a direction cosine is: $\dot{\mathbf{c}}_i = S(\mathbf{c}_i)\boldsymbol{\omega}_{ob}^b$. Using these relationships Equation 4.44 can be rewritten to:

$$\dot{V} = (\boldsymbol{\omega}_{ob}^b)^T \left(\omega_o \mathbf{I} \times \mathbf{c}_2 \boldsymbol{\omega}_{ob}^b + \omega_o \mathbf{c}_2 \times (\mathbf{I}\boldsymbol{\omega}_{ob}^b) - \omega_o^2 \mathbf{c}_2 \times \mathbf{I}\mathbf{c}_2 + 3\omega_o^2 \mathbf{c}_3 \times \mathbf{I}\mathbf{c}_3 + \boldsymbol{\tau}^b \right) + 3\omega_o^2 \mathbf{c}_3^T I \dot{\mathbf{c}}_3 - \omega_o^2 \mathbf{c}_2^T I \dot{\mathbf{c}}_2 \quad (4.45)$$

Since $\dot{V}$ is a scalar the terms in $\dot{V}$ can be transposed, leading to all terms canceling each other out except the torque. The Lyapunov derivative can be written as:

$$\dot{V} = (\boldsymbol{\omega}_{ob}^b)^T \boldsymbol{\tau}^b \quad (4.46)$$

Chetaev's theorem, [23] and [24], can then be used to prove instability by showing that:

$$V(0) = 0 \quad \text{and} \quad V(x) > 0, \quad \forall x \neq 0$$
$$\dot{V}(x,t) > 0 \quad \forall x \neq 0$$

To prove instability the torque expression for the tumbling controller is inserted in Equation 4.46, this expression is:

$$\boldsymbol{\tau}^b = h \frac{1}{|\mathbf{B}^b|^2} (\mathbf{B}^b \times \boldsymbol{\omega}_{ob}^b) \times \mathbf{B}^b, \quad h > 0 \quad (4.47)$$

Inserting Equation 4.47 into Equation 4.46 the Lyapunov derivative can be written as:

$$\begin{aligned} \dot{V} &= h \frac{1}{|\mathbf{B}^b|^2} (\boldsymbol{\omega}_{ob}^b) \left((\mathbf{B}^b \times \boldsymbol{\omega}_{ob}^b) \times \mathbf{B}^b \right) \\ &= h \frac{1}{|\mathbf{B}^b|^2} (\mathbf{B}^b \times \boldsymbol{\omega}_{ob}^b)^T (\mathbf{B}^b \times \boldsymbol{\omega}_{ob}^b) \end{aligned} \quad (4.48)$$

$\mathbf{B}^b$ will vary with time as well as the orientation of the satellite, thus $\dot{V}$ has to be considered as a function of both the state vector $\mathbf{x}$, and time, that is to say that the Lyapunov derivative

is non-autonomous, $\dot{V}(\mathbf{x}, t)$. The magnetic field vector will never be zero, $\mathbf{B}^b \neq 0$, so Equation 4.48 will only equal zero for $\boldsymbol{\omega}_{ob}^b = 0$ or when $\boldsymbol{\omega}_{ob}^b$ is parallel to $\mathbf{B}^b$. Otherwise $\dot{V}(\mathbf{x}, t) > 0$ is true.

Intuitively this will never be a problem for the tumbling controller. Irregularities in the magnetic field will ensure that even if $\boldsymbol{\omega}_{ob}^b$ is parallel to $\mathbf{B}^b$ they will diverge. The solution where $\boldsymbol{\omega}_{ob}^b$ is parallel to $\mathbf{B}^b$ is unstable, a small perturbation will cause $\boldsymbol{\omega}_{ob}^b$ to diverge from $\mathbf{B}^b$. Thus this will not be a stationary solution to the system and we can conclude that the controller is unstable.

The tumbling controller in Equation 4.25 is unstable by Chetaev's theorem [23], [24]. Energy will be added to the system and the angular velocity will increase.

4.5 Pulse Width Modulation and Controller Timing

PWM is a technique used to modulate a signal so that a square control output can be used to emulate a signal of any form, an example can be seen in Figure 4.1.

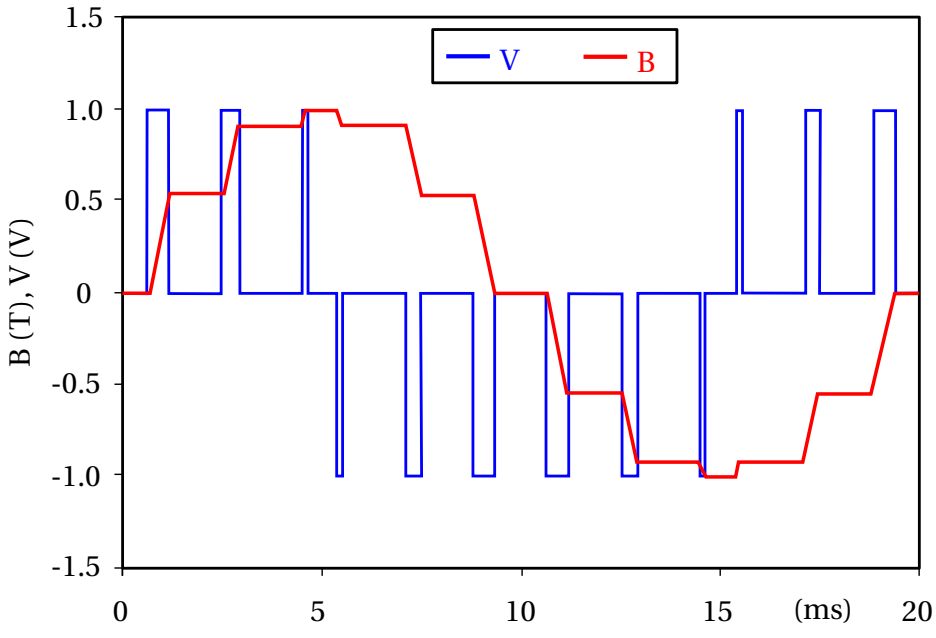

Figure 4.1: Figure showing an example of PWM in an idealized inductor, taken from [25]

The actuator design on the ADCS uses a one cycle PWM approach, Figure 4.2 shows how the control output is controlled and what it look likes.

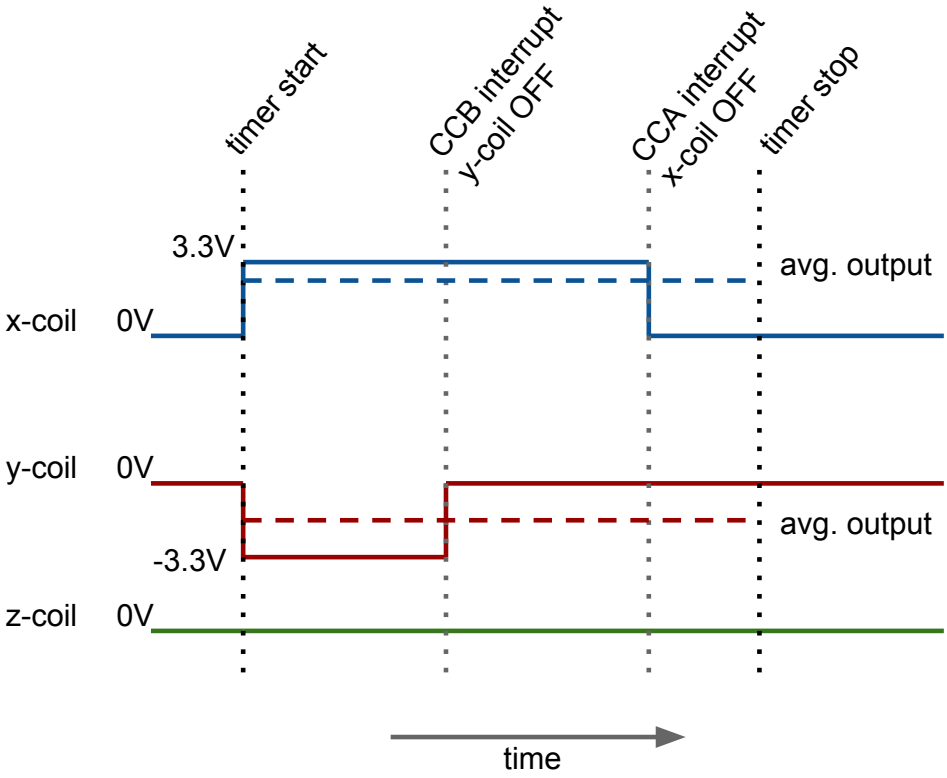

Figure 4.2: Figure showing coil output on NUTS, taken from [10]

For the rest of this thesis the period between timer start and timer stop will be defined as $T_{control}$. For code simplicity and to make the system more intuitive the control period is set to be equal to the period that is needed to perform the necessary operations to find the control output, like reading sensor data and estimating the satellites attitude. Simulations will be done in Chapter 5 to see whether or not the controllers will work with this approach.

Unfortunately due to the lack of hardware and some subsystems no accurate estimate exists on how long it will take to calculate the control output. In [9] the estimator algorithm used about 200 milliseconds, this will most likely be the slowest part of the code, especially when the estimation algorithm is updated to include a estimate of the angular velocity.

The timing of everything else is unknown, but to test the system, and as an assumption

for the simulations, $T_{control}$ is being set to 500 milliseconds. The total period will then be 1 second. The control output can also be calculated while the coils are running (the magnetometers can not be used while the coils are running though), but the ADCS will also need to have time to perform other house keeping operations like message handling.

Another factor to consider for the simulations is the time lag that exists in an inductor, like the magnetorquers, after a current has started to run trough it until the magnetic field have reached its maximum value. Table 3.1 shows that the time constants for the magnetorquers are $\tau_{x,y} = 313\mu s$ and $\tau_z = 494\mu s$. The time constant is the time it takes for the current in an inductor to reach 63.2% of its maximal value, five time constants is considered as the time it takes for the magnetic field produced by an inductor to reach its maximum value. For the magnetorqers five time constants are:

$$5\tau_{x,y} = 1.565ms \tag{4.49}$$

$$5\tau_z = 2.4ms \tag{4.50}$$

This is considerably shorter than $T_{control}$, and the time it takes to charge and discharge the magnetorquers are therefore ignored in the simulations.

To summarize, the system will first run for a period of $T_{control}$. At the start of this period the data needed to calculate the control output will be stored and calculations can begin. When this period is over the control output should be finished calculating and the magnetorquers can be turned on, the magnetorquers then stay on for a maximum period of $T_{control}$. While they are on they will constantly produce the maximal magnetic dipole moment. The magnetor-

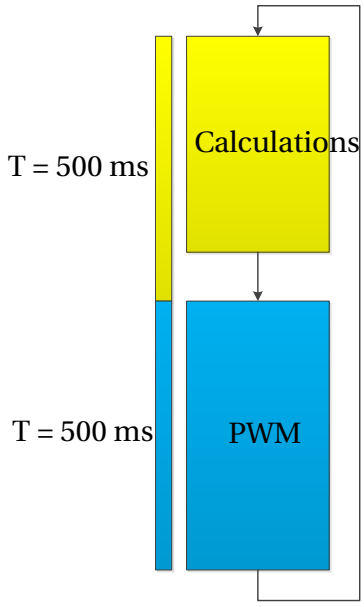

Figure 4.3: Figure showing the system flow

quers might be turned off earlier by an interrupt. How long they stay on is found by comparing the value found by the control algorithms to the maximal value that can be set up by the magnetorquers. This will then be repeated as long as the controller is active, as can be seen in Figure 4.3 in which $T_{control} = T$. The process is further mimicked in the simulations in Chapter 5. This assumption have also be used when designing the software.

Chapter 5

Simulations

Parts of the introduction to this chapter have been taken from [5], all of the simulation results are new. In this chapter numerical simulations done in Matlab will be presented for the different controllers. The code used is based on that of [5], but edited so that the sensor data is stored periodically and the control output is used according to the one cycle PWM method. For the calculations from quaternions to Euler angles and back the Marine Systems Simulator (MSS) toolbox is being used. The toolbox has been developed for control and simulation of marine craft, but the functions for dealing with quaternions, Euler angles and rotation matrices are very useful for any application that are operating with these parameters. For more on the MSS toolbox see [26]. Matlabs built in $ode45$ function is being used to simulate the system, which is given by the state vector:

$$\mathbf{x} = \begin{bmatrix} (\boldsymbol{\omega}_{ib}^{b})^T & \eta & \boldsymbol{\epsilon}^T & \lambda & W \end{bmatrix}^T \tag{5.1}$$

This is the state vector first used in [6] and the one that has been used throughout the entire NUTS project. Note that it is not the same state vector that was used for the instability evidence in Chapter 4

$\boldsymbol{\omega}_{ib}^{b}$ is the angular velocity of the satellite in the body frame with respect to the ECI frame, given in the body frame. η is the first element of the unit quaternion vector, $\boldsymbol{\epsilon}$ consists of the three last elements of the unit quaternion, λ is the latitude and W is the satellites power consumption. W does not have any effect on the satellites orientation or control, but is included to monitor the power usage. It is not being used in this thesis.

The initial conditions are set in $\boldsymbol{\omega}_{ob}^{b}$ (in radians per second), and Euler angles $[\phi\,\theta\,\psi]$ (in degrees), which are the attitude variables of the satellite in the body frame with respect to the orbit frame, and then transformed to the variables in the state matrix using the MSS toolbox. The initial latitude, λ, and power consumption, W, is set to zero in all the simulations. The latitude, λ, is included since it is needed for the International Geomagnetic Reference Field (IGRF) model that is being used to calculate the magnetic field [3].

The satellite parameters can be found in Table 5.1. Since the satellite has not been built yet all these variables are based on assumptions and approximations. The inertia are calculated assuming uniform mass distribution, then changed slightly to satisfy the stability requirement: $I_y > I_x > I_z$.

Table 5.1: Satellite Parameters

Parameter	**Value**
m [kg]	2.6
I_x $[kgm^2]$	0.0098
I_y $[kgm^2]$	0.0108
I_z $[kgm^2]$	0.0043
Voltage [V]	3.3
Rs[km] (orbit height)	600
Orbit Period [min]	96.54

5.1 Detumbling

Here the simulation results for the detumbling controller are presented. The simulations have been done for two different controller periods to see how much the period will affect the effectiveness of the controller.

$$T_{control1} = 0.25 \quad \text{seconds} \tag{5.2}$$

$$T_{control2} = 0.5 \quad \text{seconds} \tag{5.3}$$

$T_{control1}$ is used for detumbling simulation 1, the initial values are defined in Table 5.2 and a plot of the angular velocity can be found in Figure 5.1. $T_{control2}$ is used for detumbling simulation 2 and 3. The initial values used for detumbling simulation 2 can be found in Table

5.3, and the resulting simulation results is presented in Figure 5.2. The initial values used for detumbling simulation 3 can be found in Table 5.4, and the resulting simulation results are presented in Figure 5.3.

Tuning lead to the following control gain being selected:

$$d = 1 \times 10^{-4}; \tag{5.4}$$

Only plots of the angular velocity is included here, since this is the most interesting result to look at for the detumbling controller.

Table 5.2: Initial Values for Detumbling Simulation 1

$\boldsymbol{\omega}_{ob}^{b} = [1.2 \quad 1.2 \quad 1.2]^T$
$\boldsymbol{\Theta} = [90 \quad -50 \quad 20]^T$
$\mathbf{q} = [0.5792 \quad 0.6830 \quad -0.1830 \quad 0.4056]^T$

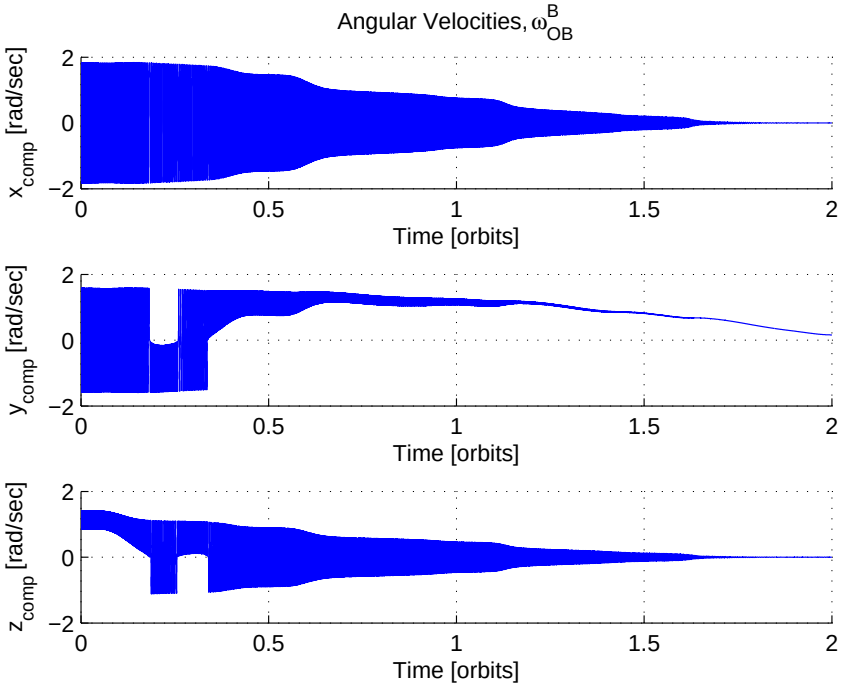

Figure 5.1: Detumbling Simulation 1, Angular Velocities

Table 5.3: Initial Values for Detumbling Simulation 2

$\boldsymbol{\omega}_{ob}^{b} = [0.9 \quad 0.9 \quad 0.9]^T$
$\boldsymbol{\Theta} = [90 \quad -50 \quad 20]^T$
$\mathbf{q} = [0.5792 \quad 0.6830 \quad -0.1830 \quad 0.4056]^T$

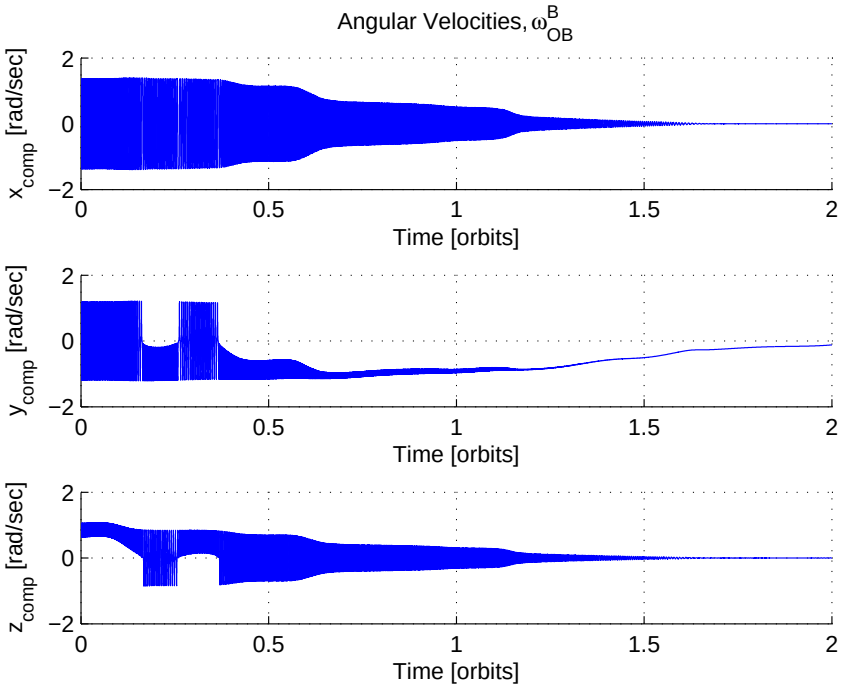

Figure 5.2: Detumbling Simulation 2, Angular Velocities

Table 5.4: Initial Values for Detumbling Simulation 3

$\boldsymbol{\omega}_{ob}^{b} = [0.1 \quad 0.1 \quad 0.1]^T$
$\boldsymbol{\Theta} = [90 \quad -50 \quad 20]^T$
$\mathbf{q} = [0.5792 \quad 0.6830 \quad -0.1830 \quad 0.4056]^T$

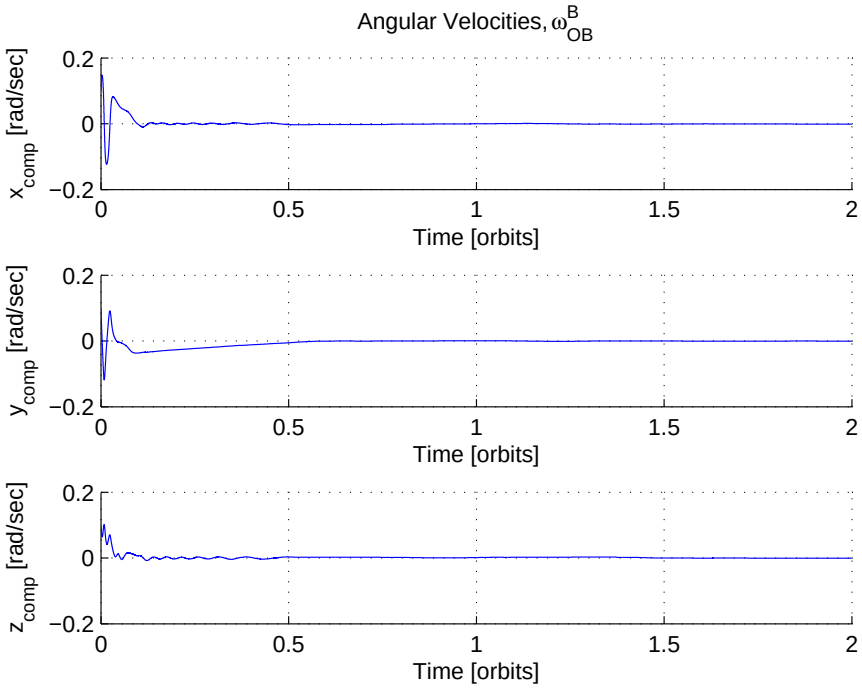

Figure 5.3: Detumbling Simulation 3, Angular Velocities

One of the main concerns when it comes to the detumbling controller is the initial angular velocity. The spin when launching from the Poly-Picosatellite Orbital Deployer (P-POD) is unknown, and no defined maximum limit has been found. In [19] there were several sources claiming that the worst initial tip-off rate was known to be 5.7 degrees per second (0.1 radians per second), without finding any information confirming this. SwissCube experienced an initial spin of about 360 degrees per second (6.28 radians per second) [27], which is the highest angular velocity reported (that the author has been able to find).

Simulations done here have found that the detumbling controller will work for speeds up to approximately 51 degrees per second, assuming that $T_{control}$ = 0.5 seconds. By reducing $T_{control}$ to 0.25 seconds the controller could handle speeds up to 69 degrees per second. This is not a significant improvement, and it might be favorable to run the detumbling controller with the same period as the rest of the system.

Experimentation with tuning of the filter parameters presented in Chapter 4.2 have also been tried. The filter coefficients in Equation 4.10 seems to affect the operating range of the detumbling controller much more than the control gain, d. The filter has been tuned in such a way that the detumbling controller could work with initial speeds around 180 degrees per second if it was run with a shorter time period. Unfortunately the filter parameters are not actually tuning variables, unlike the control gain. As can be seen in Chapter 4.2 they have fixed expressions that are dependent on the cutoff frequency, the passband gain and the sampling time. The only tunable parameter is the passband gain, and varying it did not have much effect. The simulations shown here have been performed with a passband gain of K = 1.

5.2 Pointing

Here the simulation results for the pointing controller are presented. Simulations have been performed both with and without measurement noise on the gyroscope measurements. After seeing how large an effect measurement noise had on the controller during hardware testing in [5] it has been desirable to find a limit on how much noise that is acceptable. For the pointing controller all the presented results have been performed with the following pe-

riod:

$$T_{control} = 0.5 \quad seconds \tag{5.5}$$

Tuning of the controller lead to the following gains being used:

$$p = 1.8 \times 10^{-6} \tag{5.6}$$

$$d = 1.4 \times 10^{-3} \tag{5.7}$$

5.2.1 Without Measurement Noise

Two simulations are being presented here with no noise on any of the measurements. The simulations are included to illustrate how much the initial conditions affects how long it takes for the controller to steer the satellite to the desired attitude.

Pointing simulation 1 illustrates the maximum convergence rate, its initial values are given in Table 5.5 and the result is presented in Figure 5.4 and Figure 5.5. Figure 5.6 and Figure 5.7 shows an example of how long it can take for the controller to stabilize the satellite (pointing simulation 2), the initial conditions are given in table 5.6.

Table 5.5: Initial Values for Pointing Simulation 1

$\boldsymbol{\omega}_{ob}^{b} = [0.002 \quad -0.03 \quad 0.013]^T$
$\boldsymbol{\Theta} = [-180 \quad -50 \quad -180]^T$
$\mathbf{q} = [-0.4226 \quad 0.0000 \quad 0.9063 \quad 0.0000]^T$

Table 5.6: Initial Values for Pointing Simulation 2

$\boldsymbol{\omega}_{ob}^{b} = [0 \quad 0 \quad 0]^T$
$\boldsymbol{\Theta} = [-180 \quad 50 \quad 70]^T$
$\mathbf{q} = [0.2424 \quad 0.7424 \quad 0.5198 \quad -0.3462]^T$

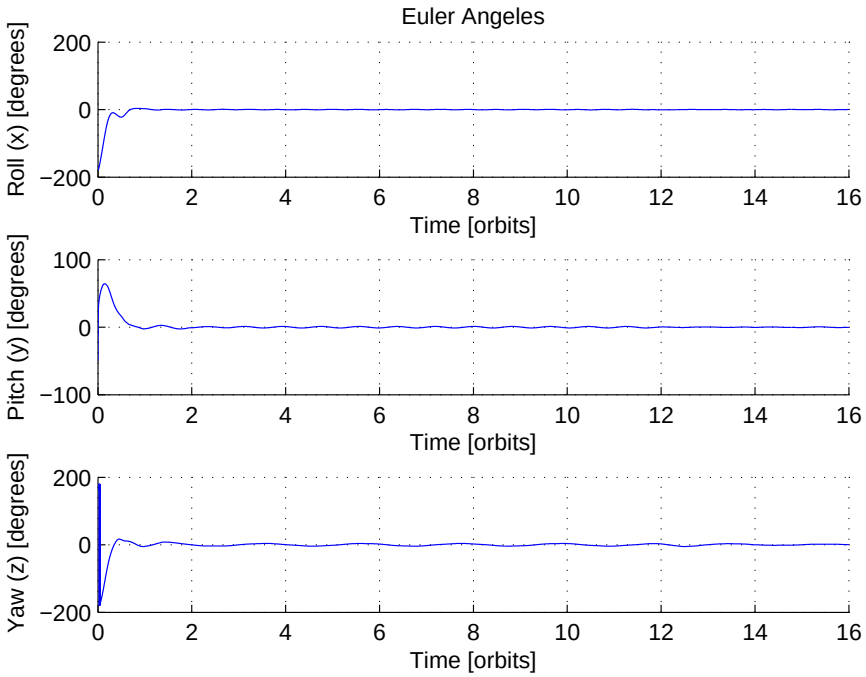

Figure 5.4: Pointing Simulation 1, Euler Angles

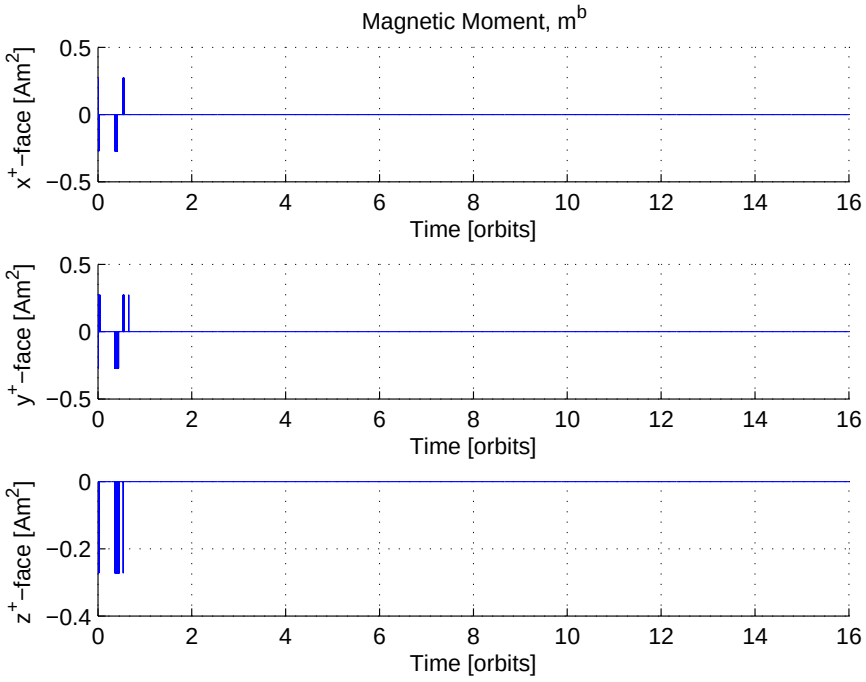

Figure 5.5: Pointing Simulation 1, Magnetic Moment

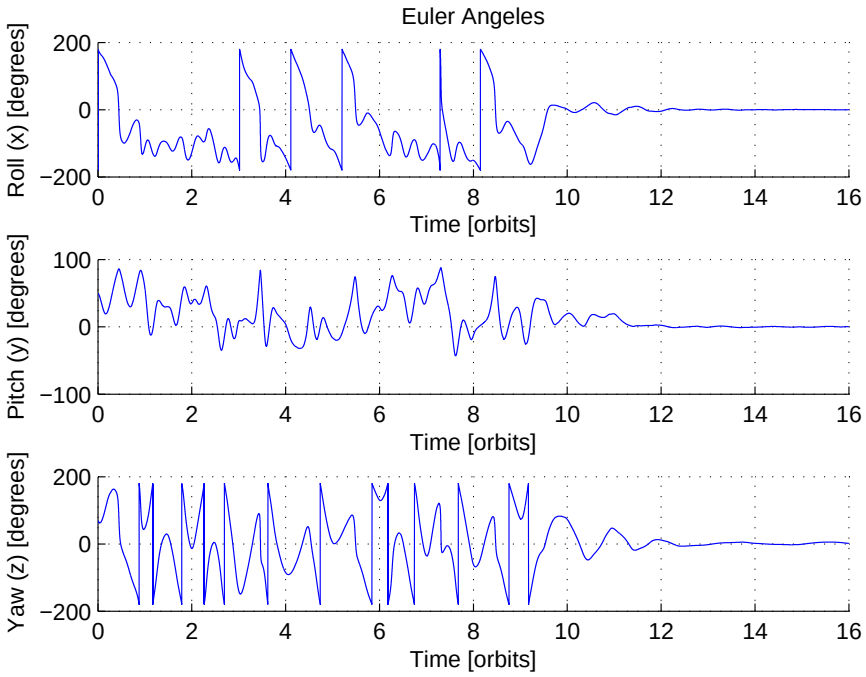

Figure 5.6: Pointing Simulation 2, Euler Angles

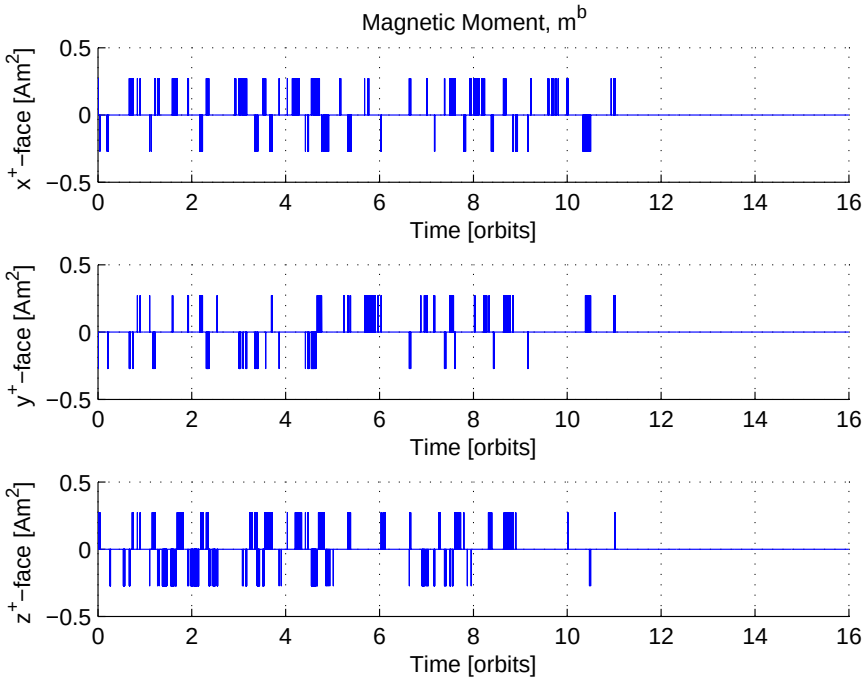

Figure 5.7: Pointing Simulation 2, Magnetic Moment

It is clear when comparing Figure 5.4 to Figure 5.6 that with the initial conditions in Table 5.5 the controller can reach the desired attitude much faster than with the initial conditions in Table 5.6. These two figures shows the two extremes that have been found, a more typical result is that it takes 4 - 8 orbits for the satellite to reach the desired attitude, instead of 1 or 12 orbits. This can be seen in Figure 5.8.

The slight wobbling that can be seen in Figure 5.4 after 2 orbits is well within the ±25 degree accuracy, as is the wobbling seen in Figure 5.6 after approximately 12 orbits. Looking at Figure 5.5 one can also see that the controller is no longer generating any magnetic moment after approximately 1 orbit, this is expected as the satellite is stable for the nadir pointing attitude without the controller. The same can be seen in Figure 5.7 after approximately 12 orbits. Different initial values for $\boldsymbol{\Theta}$ have been found to be the main factor for how long it takes for the satellite to stabilize in the correct attitude.

The pointing controller is only being used to push the satellite into the stable attitude where the body frame is aligned with the orbit frame, once the satellite has reached that attitude it should not require any more running of the magnetorquers. It is important to note that these simulations have been run with the gravity gradient as the only external torque, in reality there will also be other forces generating torque on the satellite that will affect the stability of the nadir pointing. These forces will be smaller than the gravity torque and the torque set up by the magnetorquers. That is to say that in reality the magnetorquers will be used also after the satellite have reached the desired attitude. The simulation results presented for the pointing controller with noise on the angular velocity measurements in the next section is thus more realistic.

5.2.2 With Measurement Noise

Three simulations are presented here to show how the pointing controller behaves with different kinds of noise on the angular velocity measurements. The same initial conditions are being used for all simulations, given in Table 5.7, the type of added noise can be found in Table 5.8. Pointing simulation 4 is without any noise to illustrate how the pointing controller should behave under ideal circumstances and can be seen in Figure 5.8, this is how the pointing controller behaves on average in the simulations. Figure 5.9 shows the Euler Angles for

the satellite when white noise is added to the angular velocity measurements (pointing simulation 5). Figure 5.12 shows the Euler Angles for the satellite when sinusoidal noise is added to the angular velocity measurements (pointing simulation 6).

Table 5.7: Initial Values for Pointing Simulation 4-6

$\boldsymbol{\omega}_{ob}^{b} = [0.002 \quad -0.03 \quad 0.013]^T$
$\boldsymbol{\Theta} = [130 \quad 50 \quad -90]^T$
$\mathbf{q} = [0.0000 \quad 0.7071 \quad -0.4545 \quad -0.5417]^T$

Table 5.8: Noise Types for Simulations

Simulation	**Noise Type**	**Amplitude/Variance**
Pointing Simulation 4	No Noise	N/A
Pointing Simulation 5	Normally distributed random numbers	Variance = 4×10^{-8}
Pointing Simulation 6	Sinusoidal	Amplitude = 2×10^{-5}

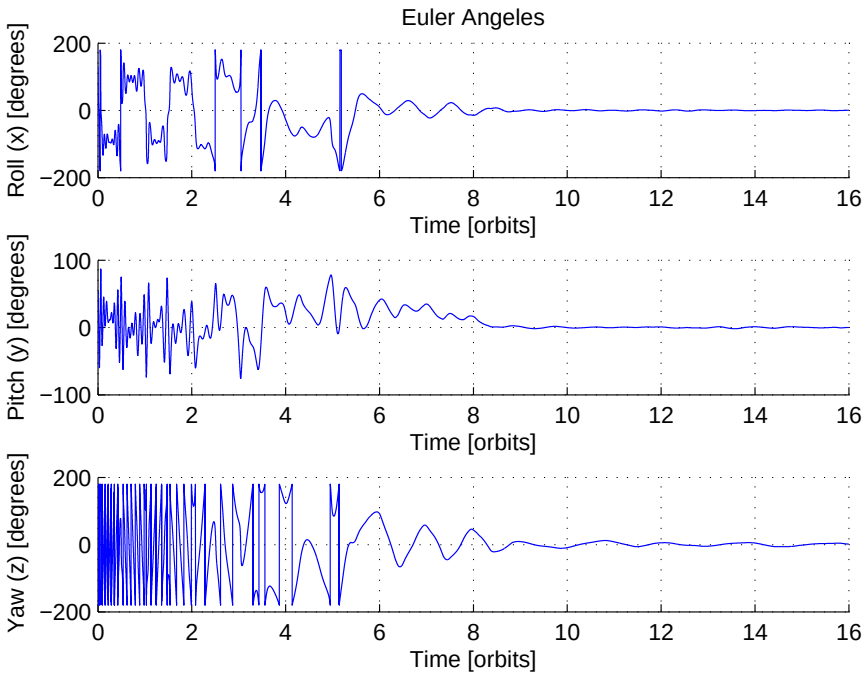

Figure 5.8: Pointing Simulation 4, Euler Angles

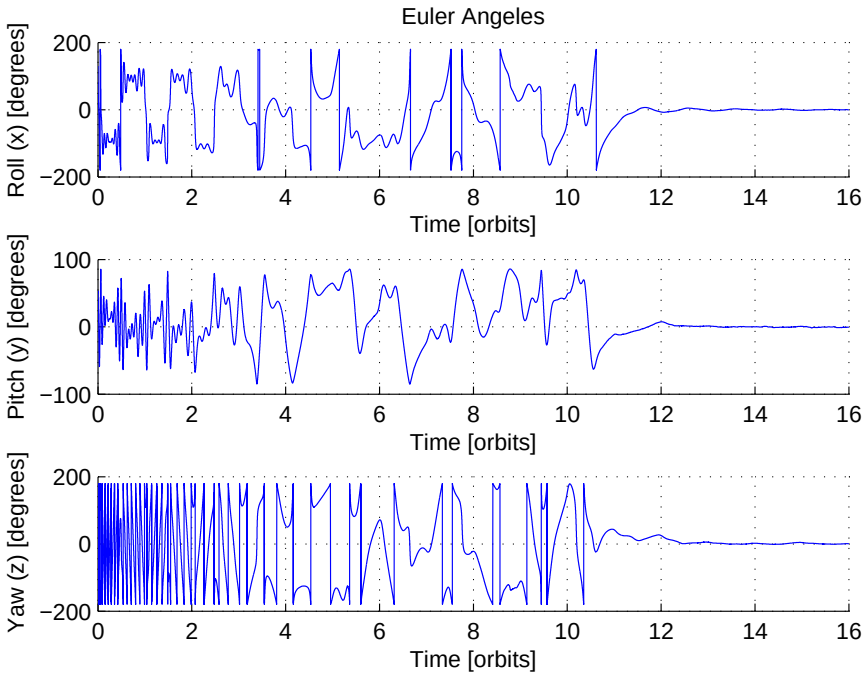

Figure 5.9: Pointing Simulation 5, Euler Angles, with white noise

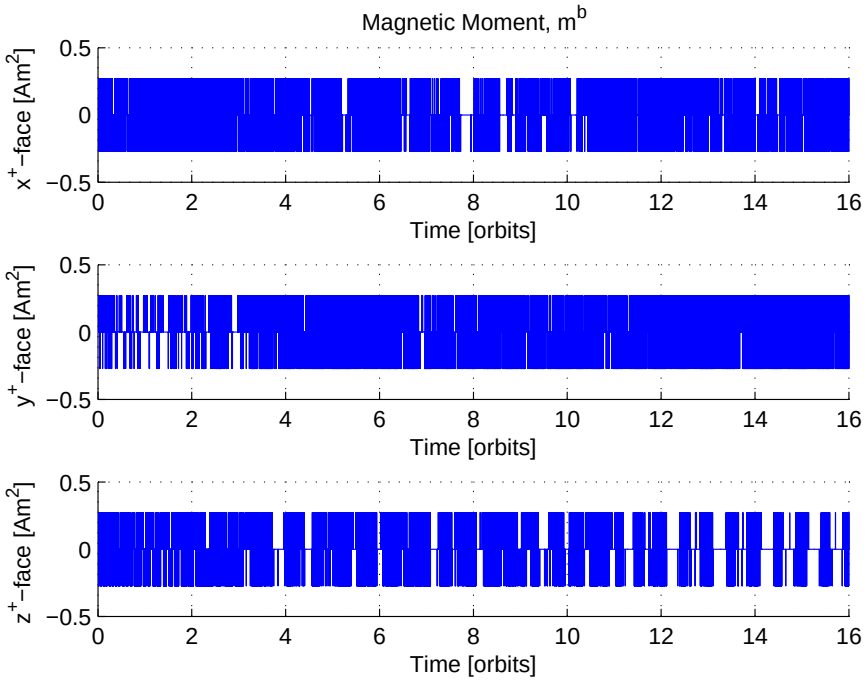

Figure 5.10: Pointing Simulation 5, Magnetic Moment, with white noise

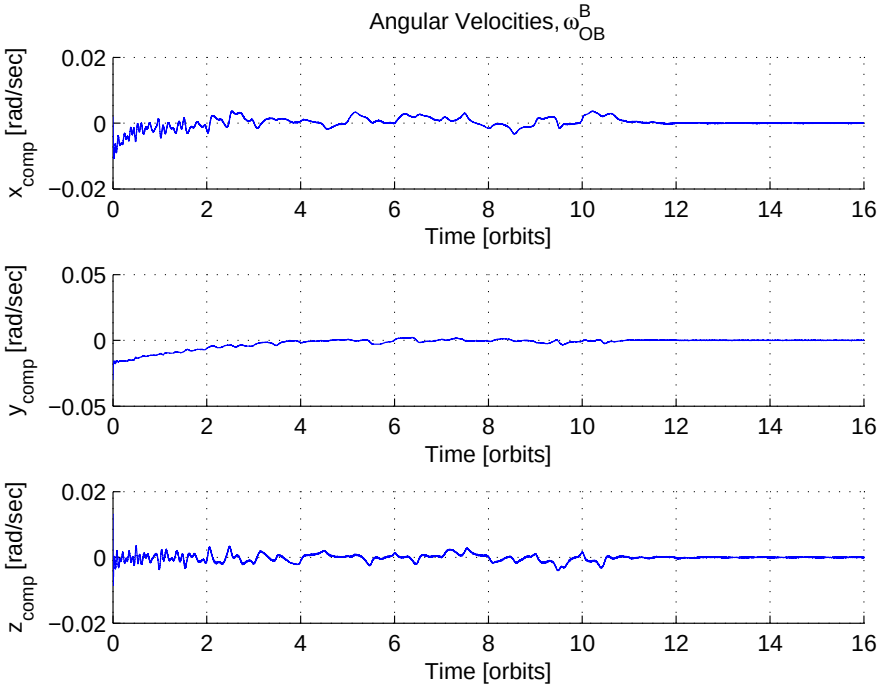

Figure 5.11: Pointing Simulation 5, Angular Velocities, with white noise

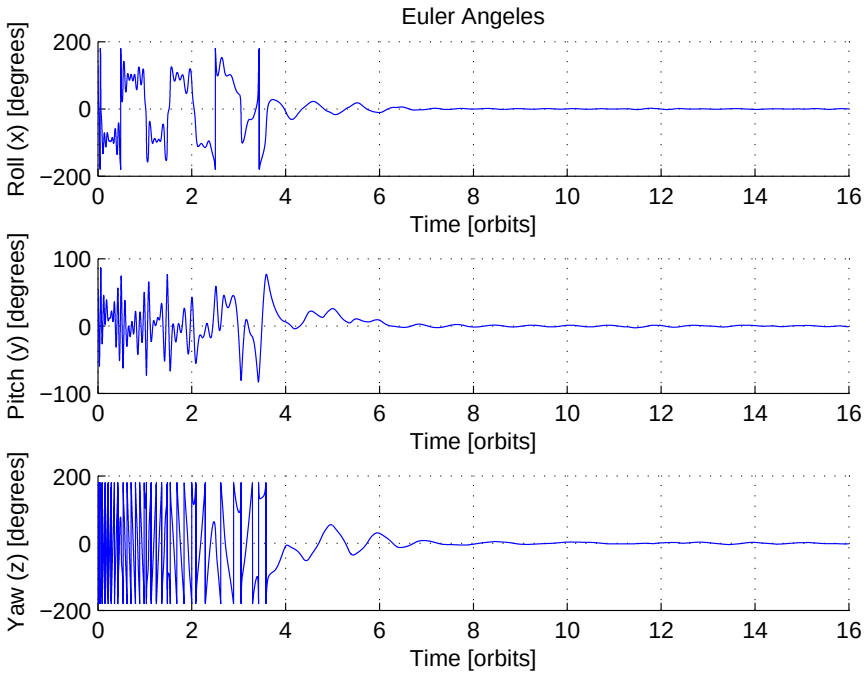

Figure 5.12: Pointing Simulation 6, Euler Angles, with sinusoidal noise

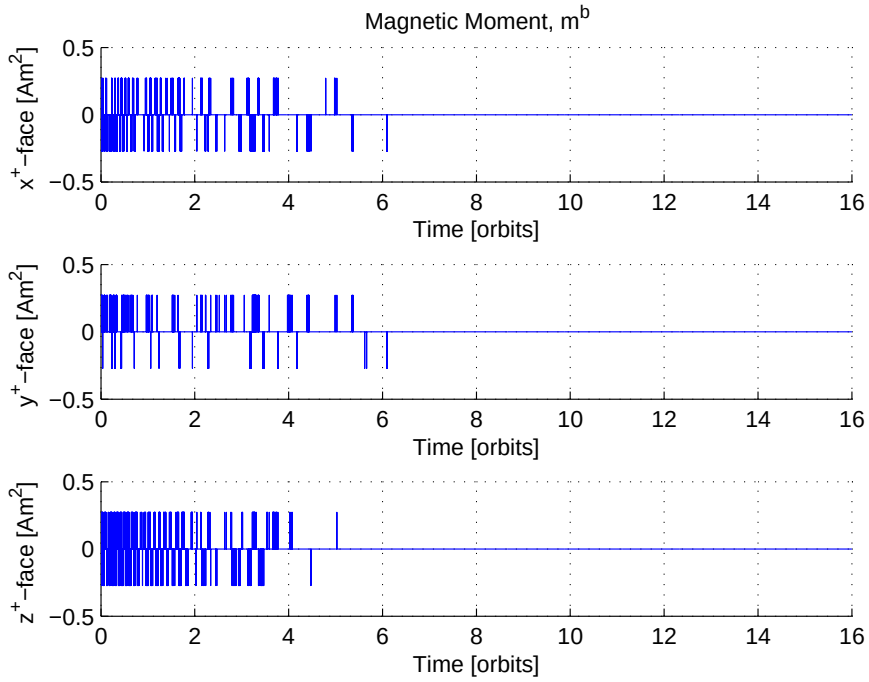

Figure 5.13: Pointing Simulation 6, Magnetic Moment, with sinusoidal noise

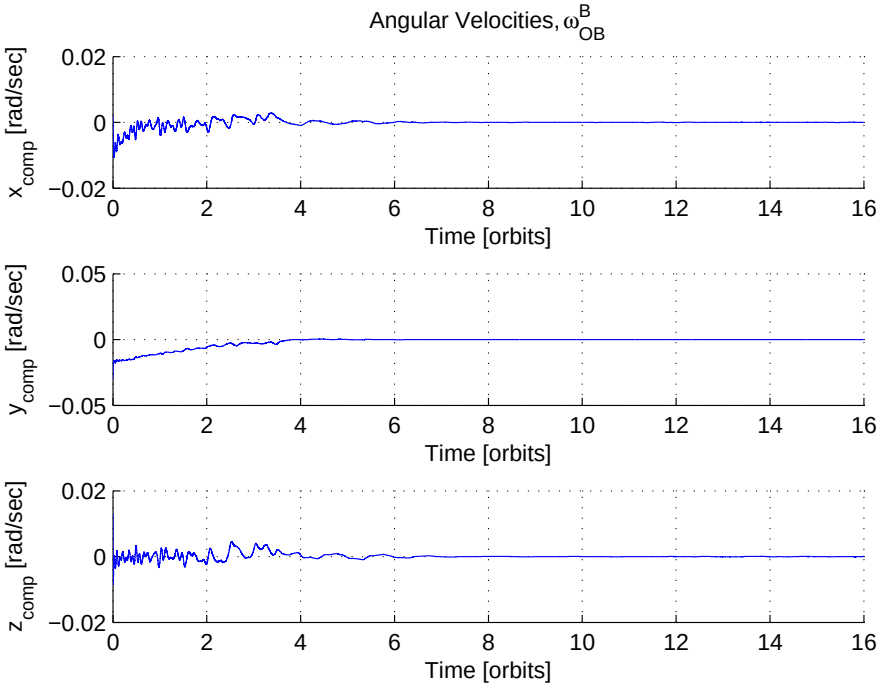

Figure 5.14: Pointing Simulation 6, Angular Velocities, with sinusoidal noise

Figure 5.10 shows the magnetic moment for pointing simulation 5, while Figure 5.9 shows the Euler angles. The angular velocity for pointing simulation 5 can be seen in Figure 5.11. Note that the controller never stops setting up a magnetic moment, unlike how it behaved in the simulations where there was no noise. This is as expected, the controller constantly sets up a magnetic moment to counteract the velocity it thinks the satellite has because of the noise. White noise with a variance of less than or equal to 4×10^{-8} on the angular velocity measurements are thus tolerable.

Figure 5.13 shows the magnetic moment for pointing simulation 5, while Figure 5.12 shows the Euler angles. The angular velocity for pointing simulation 5 can be seen in Figure 5.14. Notice here that the controller stops setting up a magnetic moment once the satellite has reached the desired attitude. This is because the noise level is so low that the controller is not activated, with higher noise levels the satellite does not reach the desired attitude. Any sinusoidal noise on the angular velocity measurements have to have an amplitude of less than 2×10^{-5} for the controller to work as expected. Any more noise and the controller becomes unstable.

This results is as expected, slowly varying sinusoidal noise is impossible for the controller to remove without having any more knowledge of it. While hardware testing the first ADCS prototype in [5] sinusoidal noise was spotted on the angular velocity measurements, thus this effect have to be removed for the pointing controller to work. White noise is what is usually used to check robustness for a controller, as can be seen the pointing controller will still work when there are white noise on the angular velocity measurements.

Simulations have only been performed for noise on the angular velocity measurements. The magnetometer chosen for NUTS have proven to be extremely stable [5], and a low pass filter for these measurements was developed in [2]. The quaternion used is a unit quaternion, that means that it will always be a unit vector, this naturally limits any noise that might occur on the estimate and has therefore not been considered.

5.3 Tumbling

Here the simulation results for the tumbling controller is presented. The control period is:

$$T_{control} = 0.5 \quad \text{seconds} \tag{5.8}$$

Tuning of the controller lead to the following gain being selected:

$$h = 1 \tag{5.9}$$

With this gain the controller goes to a maximum output immediately, which simulations show is the most efficient way to spin up the satellite. The gain could also be set lower, but by setting it to 1 it can be ignored in the software implementation. Only a plot of the angular velocity is included here since it is the most interesting result.

Figure 5.15 shows the angular velocity for the satellite without any control input for one orbit, then the tumbling controller is turned on and the satellite starts spinning. A small disturbance is used to push the satellite out of the equilibrium point. The initial values are given in Table 5.9.

Table 5.9: Initial Values for Tumbling Simulation

$\boldsymbol{\omega}_{ob}^{b} = [0 \quad 0 \quad 0]^T$
$\boldsymbol{\Theta} = [0 \quad 0 \quad 0]^T$
$\mathbf{q} = [1 \quad 0 \quad 0 \quad 0]^T$

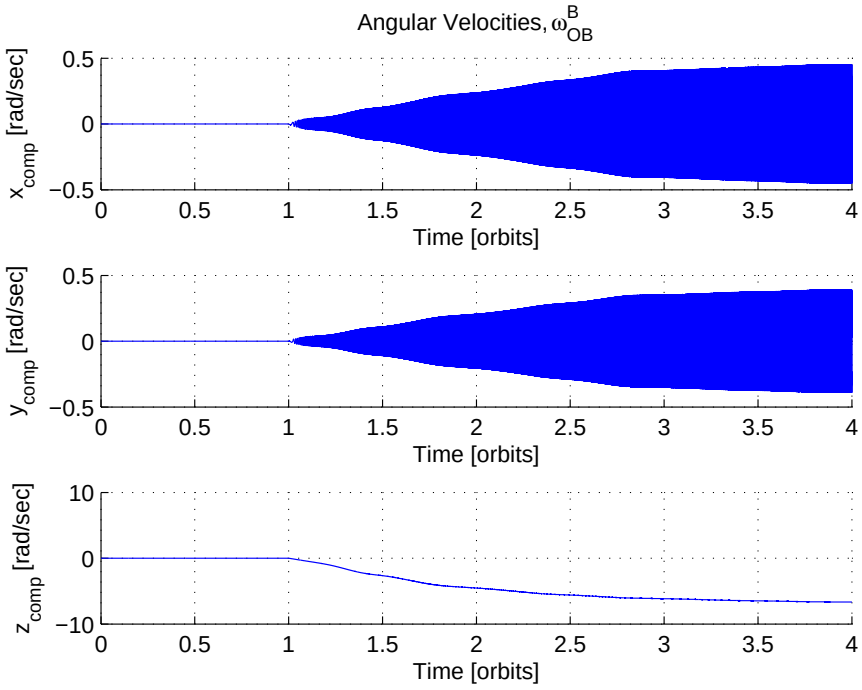

Figure 5.15: Tumbling Simulation, Angular Velocity

Simulations have also been performed with two other tumbling controllers. One where the detumbling controller was used as a tumbling controller by changing the controller sign, another one where the angular velocity used in the control expression was taken directly from the gyroscope, $\boldsymbol{\omega}_{ib}^{b}$. The simulation presented here has been performed with the tumbling controller given in Chapter 4. This controller was the most effective in the simulations, and had the most rigid instability proof, and was therefore chosen. Figure 5.15 shows that the controller will in fact be able to increase the angular velocity of the satellite. It is important that the tumbling controller is not run so long that the angular velocity of the satellite exceeds what the detumbling controller can stop.

Chapter 6

Hardware

The design of the second ADCS prototype is based on the first prototype developed in [10]. Many of the designs and components have stayed the same, the main difference is that more sensors have been added for redundancy, and the microcontroller has been changed. This chapter will cover the hardware being used on the second prototype. First the changes that have been made to the hardware will be presented, then the new design. The hardware has been designed together with Henrik Ruudi Haave, and the implementation in Altium Designer has been done in cooperation with Amund Gjersvik and Henrik Ruudi Haave.

6.1 Sensors

It was early on decided that the ADCS should have more sensors [9]. Having two or more sensors adds a layer of redundancy to the system, meaning that if one sensor should malfunction another one can provide the necessary data and normal operation can continue. Several algorithms exists to handle the sensor redundancy in software, this is further investigated in Chapter 8.

6.1.1 Magnetometers

The magnetometers are going to be the same as on the first ADCS prototype [10], the HMC5983 from Honeywell [28]. This looks to be one of the better sensors out there and have worked excellently during testing [5, 9, 19]. It also has the advantage of having built in temperature compensation, which will make it more stable for the wast range of temperatures in space.

Three magnetometers have been placed on the ADCS prototype board, with the possibility of adding three more in the center of the coils (or anywhere else on the satellite) at a later stage through pins mounted on the board. External magnetometers might be useful if it turns out that the magnetic disturbance from the rest of the satellite will be a problem when the magnetometers are mounted on the ADCS board. They can also be used to measure the coil output if they are placed in the middle of the magnetorquers. The exact mounting location for any external magnetometers will need to be decided after the satellite itself have been designed and built.

The addition of more magnetometers is possible trough a demultiplexer placed on the board, it is set to operate in a 2:3 way, meaning that two input signals controls three output signals. The input signals are two Serial Peripheral Interface (SPI) chip select signals that thus can be used to generate three SPI chip select signals.

6.1.2 Gyroscopes

The gyroscopes selected are the MAX21000 from Maxim Integrated [29], this model has been chosen because of its low operating range. MAX21000 can measure speeds as low as ±0.001 degrees per second, while the gyroscope on the original prototype [10] could measure speeds as low as ±0.006 degrees per second [30]. For perfect pointing at a orbit altitude of 600 km the satellite will rotate with approximately 0.66 degrees per second. Even tough both gyroscopes can measure speeds of that magnitude, a better accuracy was desired.

MAX21000 also has a embedded First In First Out (FIFO) that can store 256 16-bit values for each output channel. This means that the gyroscope can record and store data, that can be accessed later. This can be used to greatly reduce gyroscope measurement noise when the satellite is rotating slowly, like it should during pointing. One can then pull a chunk of data from the gyroscopes FIFO buffer and take the average of that data to find a better value. It is important when using the FIFO buffer in an asynchronous mode, which is how it will be used, that the gyroscope records data at least 10 times faster then the microcontroller pulls data from it.

6.1.3 Sun Sensors

For the sun sensors, the operational amplifiers and the hardware design from the first prototype [10] will be used, but the photo diodes will be changed. Henrik Rudi Haave has been responsible for the sun sensors, and a full breakdown of the new photo diodes and the sun sensors will be provided in his master thesis.

6.2 Microcontroller

The ADCS has to be able to run an operating system to implement a version of Cubesat Space Protocol (CSP), this is the communications protocol that has been selected for NUTS. The satellites OBC and radio will use the AVR32 UC3A3256 from Atmel. To be able to reuse code and have the whole software system be more maintainable and uniform a microcontroller from the AVR32 UC3 family is desirable for the ADCS as well.

The AVR32 UC3C1512C from Atmel has been chosen [31], this is one of the few microcontrollers that has enough Analog-to-Digital-Converter (ADC) inputs to support the amount of sun sensors needed. It also contains a floating point unit, which will speed up the calculation of many of the algorithms used on the ADCS. All 16 ADC channels, as well as the two available SPI channels are being used to connect sensors to the microcontroller. There are also several other devices connected to other input and output pins.

6.3 Clock

In addition to the already mentioned hardware changes the second prototype also has a external Real-Time Clock (RTC). The advantage of having a external RTC is that a soft reset of the microcontroller can be performed without the satellite losing its time and date information since it can be stored in the RTC. A RTC from AVX [32] has been chosen.

This oscillator is high frequency stable for temperatures between -40^oC and 85^oC with a drift of $\pm 5ppm$. The RTC needs to be very accurate for the predictor algorithms developed for the ADCS by Antoine François Xavier Pignède [11]. It has been found that errors in time over 1 minute will lead to large errors for the prediction of the position of the satellite, and

a error below 30 seconds is preferable. With a drift of $\pm 5ppm$ it will take the satellite 69.45 days until the clock error have reached 30 seconds from the initial time. It is important to note that there will probably also be a error in the initial time used by the satellite caused by a non-constant delay time when uploading a initial time to the satellite. The drift rate can also be affected by the hard radiation operating environment.

6.4 ADCS Prototype

Figure 6.1 shows the different hardware modules mounted on the ADCS board, and how they communicate with each other. The design is similar to the one from [10], but it have been expanded upon in the ways mentioned above. The ADCS will be placed in a slave slot on the backplane. The different components are:

- Microcontroller, AVR32 UC3C1512C
- 3 X Gyroscope, MAX21000
- 3 X Magnetometer, HMC5983
- 16 X Sun Sensors (analog amplification circuits on the board)
- 3 X Coil Drivers (H-bridges on the board)
- Real Time Clock
- Demultiplexer

The connectors available are:

- Inter-Integrated Circuit (I^2C) (backplane slave connector)
- Joint Test Action Group (JTAG) (backplane and external)
- Universal Synchronous/Asynchronous Receiver/Transmitter (USART) (for bluetooth)
- Coil Connectors
- Sun Sensor Connectors
- External Magnetometer Connectors

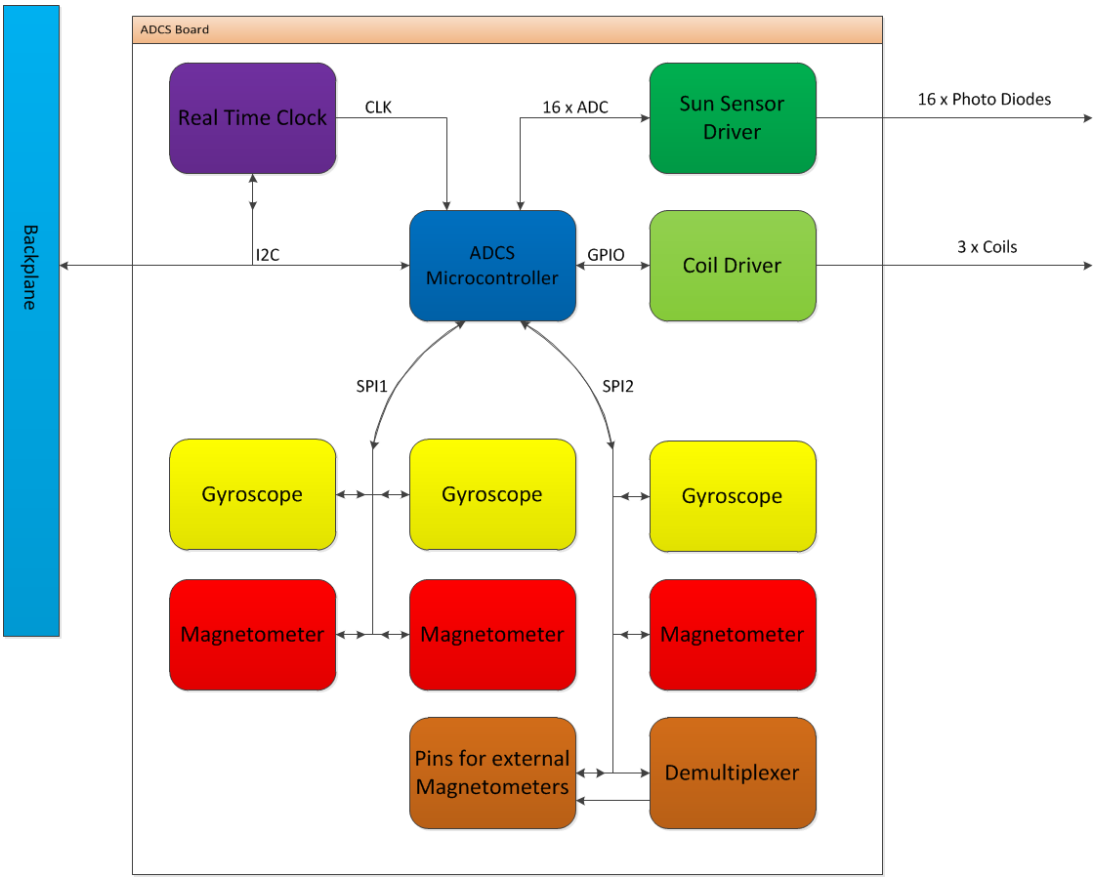

Figure 6.1: Figure showing the different hardware modules on the ADCS

The magnetometers have been placed as far away from the backplane as possible, since the electromagnetic noise will probably be smallest there. The gyroscopes have been placed as close to the center of the satellite (not the center of the board) as possible, this will provide the most accurate measurements of the angular velocity. The connectors on the board are of a type that are lockable and resistant to shaking, shock and thermal cycling, this should prevent any of the connectors falling out at any point [33].

The board will run on 3.3 Volts, which is provided from the EPS trough the backplane. The OBC can turn the ADCS board on and off, and reprogram it. The board itself measures 90mm x 92mm and is a four layer board with a earth plane everywhere where there are no signal paths on the top or bottom levels. Figure 6.2 shows a picture of the new board taken from Altium Designer. A coordinate frame is given on the board, and has been defined so that it coincides with the body frame. All of the sensors are mounted so that the data they provide is given in the body frame.

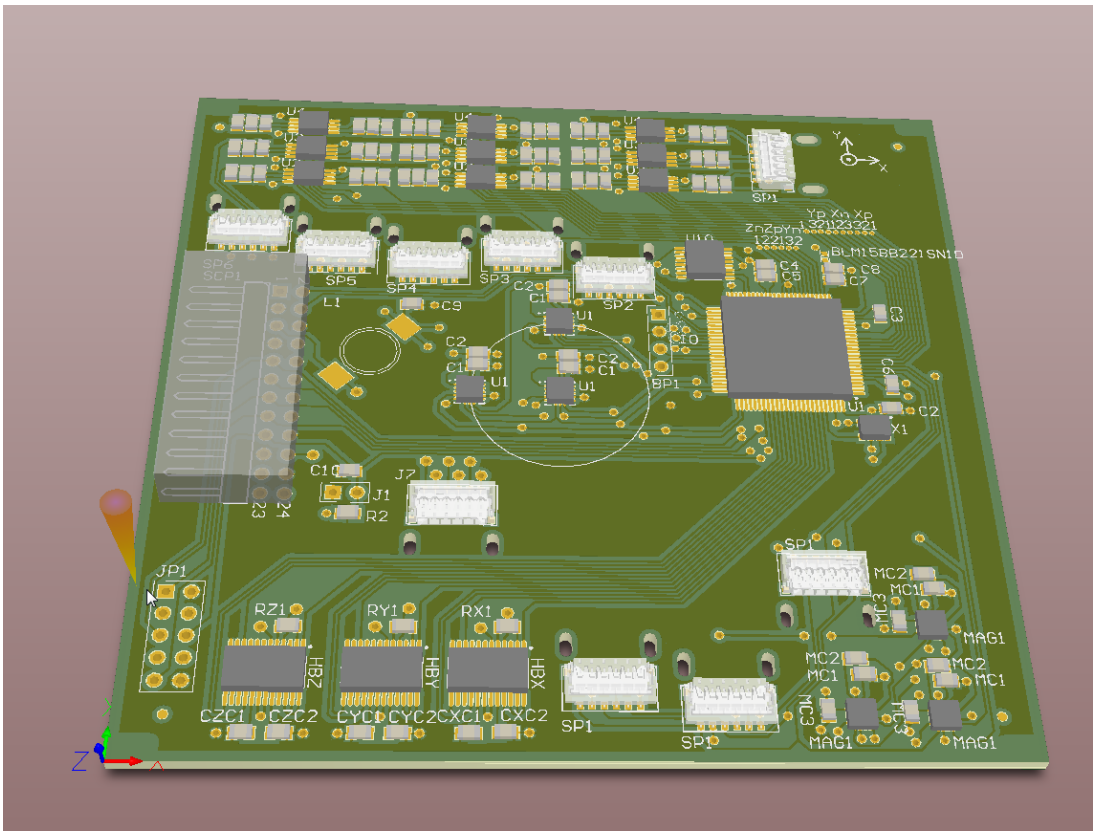

Figure 6.2: Figure showing the second prototype board for the ADCS

Chapter 7

Software

This chapter details the software architecture designed for the ADCS. First some key aspects of the software will be presented, then the design.

7.1 Operating System

The first choice that has to be made when designing software like this is whether or not the microcontroller should run an operating system. The original ADCS prototype had no operating system and only worked by using a simple state machine [10]. A goal of this second prototype has been to make a system as close to flight ready as possible, and it is thus more complex. The system can be simplified by using an operating system to mask some of the complexity and perform much of the required overhead.

A operating system called Free Real Time Operating System (FreeRTOS) has been used for software development for the radio and OBC for NUTS. It have also been used for systems on the AAUSAT3 [34] and other CubeSats. A version of CSP has been chosen for communication internally on the satellite that requires an operating system like FreeRTOS or Posix to run. FreeRTOS has therefore been chosen as the operating system for the ADCS.

FreeRTOS is a free to use real time operating system (as the name implies) [35]. The FreeRTOS home page claims that it is the market leading real time operating system. FreeRTOS is also open source and smaller than the most commonly used Linux embedded systems, making it ideal for a project like NUTS.

7.2 FreeRTOS Scheduling

FreeRTOS allows for multitasking on the microcontroller. The algorithm used is a priority based preemptive scheduling algorithm. This means that each task created needs to be assigned a priority number, the task can then either be set to run periodically, or run when a certain interrupt happens. With this algorithm the task with the highest priority will always be allowed to run. If a task with a lower priority is running when a high priority task is set to start, the high priority task will run instead of the low priority task. The low priority task will then keep running from where it was aborted after the high priority task has finished.

7.3 FreeRTOS Tasks and Task Communication

FreeRTOS works by running functions as tasks. Tasks need to be defined with a priority for the scheduler and a set amount of stack size. They can also be defined with a name and a pointer to pass other data to the task. Tasks can either be in a state of running, ready, blocked or suspended and only one task can run at a time, this is controlled by the scheduler, as already discussed.

Tasks can communicate with each other using queues. A queue in FreeRTOS can hold a fixed number of fixed sized data items and behaves like a FIFO buffer. Several tasks can be readers and writers for each queue, but there is no point in having the same task be both. The other way tasks can communicate is by using global variables. Global variables are usually considered to be less safe than message sending (queues in the FreeRTOS case), but they provide less overhead and are simpler to implement.

When using global variables it is important to make sure that only one task reads or writes to the variable at a given time, and that the read or write is completed before another task requires the variable. Deadlocks are more likely to arise with the use of global variables, as is the likelihood of the program stalling and race conditions. There does exists ways to get around these problems, the most noteworthy are semaphores and mutexes, which FreeRTOS have built in functionality for. Both are data types that are being used to control access to resources that are needed by multiple tasks so that only one task can access the resource at a given time. The main difference in FreeRTOS between the two is that mutexes inherits

the priority of any task that obtains them, while semaphores does not.

7.4 Communication

As mentioned the internal communication protocol chosen for NUTS is a version of the CSP. CSP is a network-layer delivery protocol that has been developed for CubeSats at Aalborg university [36]. A version has been made especially for NUTS this semester in a parallel project called NUTS Reliable Protocol (NRP) [37]. The NRP packages are being sent over the I^2C bus on the satellite. A suggestion for which messages the ADCS should be able to receive and send is attached in Appendix B. This list will probably be expanded upon as NUTS nears completion.

7.5 Persistent Variables

Several things can happen that causes the ADCS to perform a reset. When this happens the ADCS will relaunch with its original settings, and some data will be lost. To ensure that no critical data is lost, some data should be stored in a special way. Data like the two line element set for the predictor and the gain matrices for the estimator should be kept. To achieve this the data can be stored in the built in flash memory on the microcontroller. The flash memory consists of four parts, a factory page, a user page, a reserved part and a flash data array. The easiest way to store the variables is to use the user page, this page is 512 bytes large, and the read and write operations to this page are completely controlled by the user written software. A list of which variables should be persistent is included in Appendix A.

The problem with this approach is that flash arrays are especially susceptible to failures caused by radiation [15]. Methods should therefore be in place to ensure that the data stored in the flash user page is valid before usage.

7.6 Logging

The ADCS has no external memory to log the data it uses, the main satellite log is going to be located on the OBC. All of the sensor data and other calculated data are thus going to be sent to the OBC for storage, where it can be retrieved by the ground station at a later stage.

To avoid sending all the data every time a new control output is calculated the data is first going to be stored in a buffer on the ADCS. This is implemented as a simple ring buffer that can hold a finite and predefined number of elements. For all of the data to be kept, the OBC needs to request the buffered data from the ADCS periodically with a period that guarantees that the buffer has not been filled up in between calls. The exact timing for this needs to be found when the ADCS is completed and properly tested. The way the data will be stored in the log is defined in Appendix A.

7.7 Software Architecture

There are several ways the software for the ADCS could be designed, and during the development process several approaches have been investigated. The ADCS itself is actually a small piece of software, and it has been important not to make things more complicated than necessary. Greater complexity leads to a larger chance of introducing errors in the system. One of the main focuses of this thesis has been on the design of the software.

The code itself consists of several sub systems of different complexity and size. The overall system architecture should mask as much as this complexity as possible, making it easy to see the program flow and control how the system behaves. The code should follow the NUTS coding standard, unfortunately this has not been done for all of the projects related to the ADCS. It is therefore important that every piece of code that is written in the future adheres to the code standard.

There are two major design decisions that have to be made, firstly how many tasks is the program going to be split into, and secondly how are the tasks going to communicate with each other. The ADCS' operating modes, or states, will be presented below, then the different subsystems will be listed before looking at how they can be split into tasks. Lastly task communication will be discussed.

7.7.1 States

The states, or operating modes of the ADCS, were decided early on by the requirements of the mission and the wishes of the project leaders. The ADCS will have five different states, it is possible to transition from each state to every other, as shown in Figure 7.1. The different states are also shown in Figure 7.1. The current state will be stored as a persistent variable, so that if the system restarts it will enter whichever state it was in before the restart. The first time the system start ups the initial state will be detumbling. To move from one state to another a command from the ground station segment is required.

The sequence diagrams for each state is attached in Appendix C. To be able to log as much data as possible all of the sensors will be used no matter which state the ADCS is in. This means that even though the state is set to detumbling, and only data from the magnetometers are being used, the gyroscopes and the sun sensors will be pulled as well. The only state that does not pull any data is the Idle state, since this functions like a power saving mode for the ADCS.

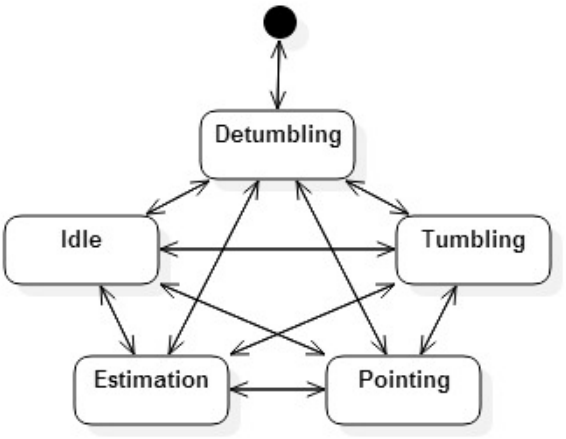

Figure 7.1: ADCS State Diagram

7.7.2 Subsystems

The ADCS consists of the following subsystems on the software level:

Magnetometer Driver

Interfaces the magnetometers to the microcontroller. Provides magnetic field measurements, 3 sets of measurements for the x, y and z axis. Software not finished since hardware was not delivered.

Gyroscope Driver

Interfaces the gyroscopes to the microcontroller. Provides angular velocity measure-

ments, 3 sets of measurements for the x, y and z axis. Software not finished since hardware was not delivered.

Sun Sensor Driver

Interfaces the sun sensors to the microcontroller. Provides intensity measurements, 16 measurements are available. Software not finished since hardware was not delivered.

Coil Driver

Interfaces the magnetorquers to the microcontroller and uses the output from the controllers to set up a current trough the magnetorquers. Software not finished since hardware was not delivered.

Magnetometer Data Processing

Calculates a magnetic field vector from the magnetic field measurements.

Gyroscope Data Processing

Calculates an angular velocity vector from the angular velocity measurements.

Sun Sensor Data Processing

Calculates a sun vector from the intensity measurements. Software not finished.

Predictor

Calculates the satellites position and velocity vectors, and provides estimates of the magnetic field vector and sun vector.

Estimator

Estimates the satellites attitude and angular velocity based on processed sensor data and predictor data. Software not finished.

Detumbling Controller

Calculates the control output for detumbling of the satellite.

Tumbling Controller

Calculates the control output for tumbling of the satellite.

Pointing Controller

Calculates the control output for pointing of the satellite.

Log

Logs the data used by the ADCS.

NUTS Reliable Protocol

Handles incoming and outgoing messages for the ADCS. Should write updated variables to the flash memory when they arrive from the ground segment.

In addition there are several overhead and standard libraries imported from the Atmel software framework that are being used in the system. This is a highly modular system, were each subsystem is a separate code part. Any one of the subsystems can be changed, as long as they adhere to the definitions of what the subsystem is supposed to do. This allows for black-box testing of each of the subsystems and makes further development and testing easier. The data structures that each subsystems uses, reads from and writes its output data to, is defined in Appendix A. A more detailed description of the software subsystems and the data storage variables can be found in the digital appendices. Functions have been made for logging and writing and reading the persistent variables to the flash memory.

7.7.3 Tasks

As mentioned earlier the system can be split into several tasks. The NRP subsystem should have its own task, but except from that the amount of tasks the rest of the system is split into is only a design decision. Two approaches have been considered.

Approach 1

In approach one there will only be two tasks, one task running the NRP subsystem and a second running a state machine. The state machine task will then use functions from all of the other subsystems to perform the required actions. The tasks would then be:

- ADCS State Machine
- NRP

With this approach the state machine itself would ensure that everything happens in the correct order according to the sequence diagrams in Appendix C. The ADCS State Machine task will run periodically every second, how often the NRP task should run have to be tested on the finished hardware.

Approach 2

The second approach is to split the system into five tasks.

- Sensors
- Predictor
- Estimator
- Controllers
- NRP

With this approach a synchronization mechanism needs to be implemented between the tasks so that every task gets to run, and is run in the proper order. That is the tasks are run in accordance with the sequence diagrams in Appendix C. FreeRTOS provides several ways to synchronize tasks like semaphores, task notification routines and event group synchronization flags. With this approach there will not be an explicit state machine, but every task will always have to know which state the system is in, and operate accordingly. The Sensor, Estimator and Controller task should be run every second, while the Predictor task can be run less frequently. Again testing have to be performed to see how often the NRP task should run.

For both approaches adding a separate task to act like an error detection and handling task could be useful, but this is not investigated further here. Figure 7.2 and Figure 7.3 shows which subsystems should belong to which tasks for the different approaches. Both of these approaches have also been implemented and tested in software, even tough the testing has been limited due to the lack of hardware.

Semaphores have been used to synchronize the tasks for **Approach 2** since the FreeRTOS manual states that semaphores are the recommended way to achieve synchronization. The author recommends **Approach 2** for the finished software, since this allows for more flexibility. Tasks can be run at different frequencies, but having more tasks adds a layer of complexity. **Approach 1** is simpler since everything will happen sequentially in the same task. This leads to less switching between tasks and makes it easier to follow the program flow. Both approaches have been tested and works, but since none of these have been tested on

the completed system both approaches are presented here. Further testing may lead to **Approach 1** being the better alternative. For both approaches it is easily possible to adjust the code so that some of the calculations are performed while the coils are being used. This could be used to run the tasks with a smaller period than one second, but it have not been tested.

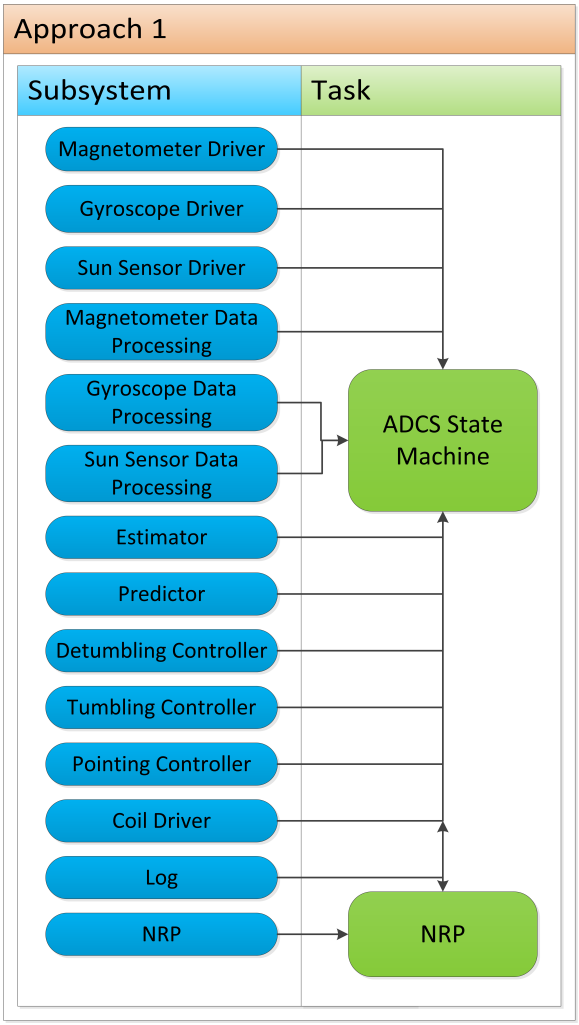

Figure 7.2: Visual representation of approach 1 of the software design

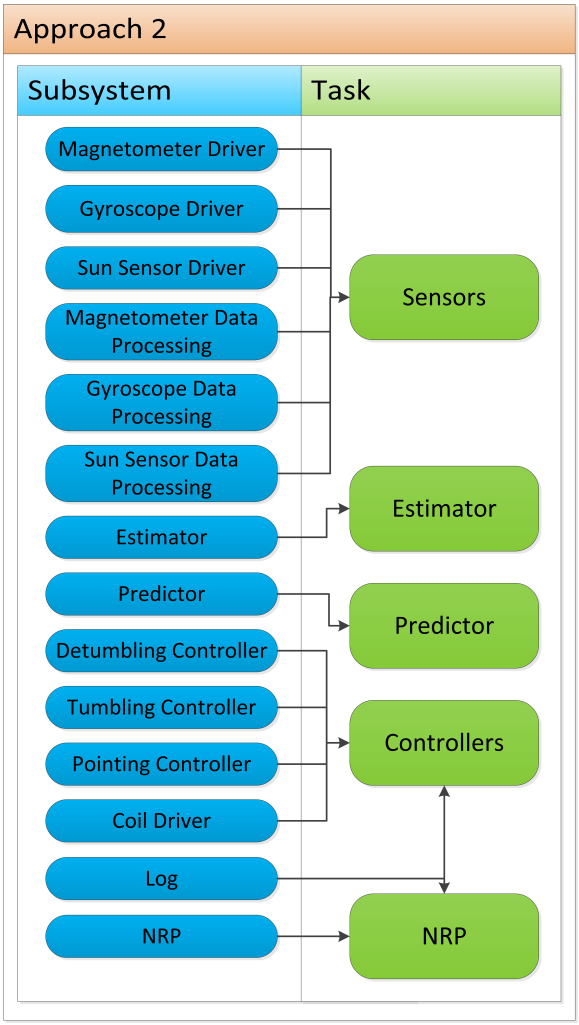

Figure 7.3: Visual representation of approach 2 of the software design

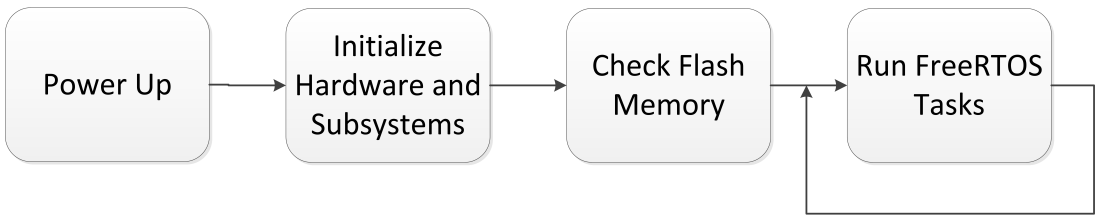

Figure 7.4: Startup Sequence

Figure 7.4 shows the ADCS startup sequence, as already mentioned the FreeRTOS tasks should be run to conform to the sequencing in Appendix C. The NRP task will run periodically, with a period that has to be defined after more testing.

7.7.4 Data Sharing

As previously mentioned, tasks can share data in two ways, either by using messages or by using global variables. Both of these approaches have been considered and tested.

Messaging has the advantage of being safer than global variables. But it does require several queues to send the messages between the tasks, introducing a large overhead to the system. It could also happen that a task will not get the time to read from a queue often enough, so that the queue will fill with by messages from the sender task, effectively blocking that queue.

The danger with global variables is the same as their advantage, they are accessible from everywhere. This means that if something goes wrong a function might change a global variable it was not supposed to. Or a situation might arise where one tasks tries to read from a variable as another task tries to write to it. Everyone who uses a global variable will be dependent on it, creating mutual dependencies and increasing complexity.

Despite all of this, global variables have been chosen for data sharing in this project. It provides less overhead and complexity for a small project such as this when compared to using messages. By using the semaphore function in FreeRTOS many of the mentioned problems is mitigated, semaphores thus have the advantage of providing task synchronization and mutual exclusion for the global variables. Global variables are thus considered to be as safe as messaging, but provides less overhead. A list of which data variables exists and which subsystems needs to access what is provided in Appendix A.

Chapter 8

Sensor Redundancy Algorithm

When a system has sensor redundancy in hardware like on the ADCS a software algorithm is required to handle the data from the sensors, and find a value to be used in the rest of the system. The ADCS will have three gyroscopes and three magnetometers, giving us redundancy on the measurements of angular velocity and magnetic field strength. This has been included since these measurements are critical for the ADCS, without them the system would not function. For the sun sensors this problem has been addressed by Henrik Rudi Haave.

The goal of the algorithm is to find the best possible value from the different values given, but finding it is not straight forward. Several algorithms exist with varying degrees of complexity and effectiveness. A study has been performed to find a suitable algorithm for the ADCS, the goal has been to find an algorithm that can function automatically and work well for three sensor while at the same time adding little complexity. The algorithm should produce a value that is as close as possible to the real value for the measurements.

Algorithms based on filter approaches and voting techniques are the most prominent in literature. Filter approaches are often more complex than voting algorithms, they require less hardware redundancy and are often used when data from different sensors are incorporated to find a measurement. For systems with hardware redundancy where the same sensors are used to find a measurement voting techniques are often used. Voting algorithms is described as the most reliable approach for a multi sensor system like the ADCS [38]. The study led to voting algorithms being chosen, and the focus of this chapter will thus be on voting

algorithms. Some background theory will now be presented, before going into specific algorithms.

Voting algorithms are split into several classes depending on input/output data and input/output votes [39], as illustrated in Figure 8.1. Exact/Inexact input data has to do with whether or not the values have to be exactly equal (exact), or just in some neighborhood of each other (inexact) to be considered the same. Consensus/Mediation output data has to do with whether or not a total agreement has to be met (consensus) for the output data, or if it can be based on a compromise (mediation).

	INPUT	OUTPUT
DATA	Exact/Inexact	Consesus/ Mediation
VOTE	Oblivious/ Adaptive	Threshold/ Plurality

Figure 8.1: Classification of voting algorithms, figure taken from [39].

In addition to the data itself, a vote can be given to the data. This can be used if for example one believes that a given sensor is more reliable than another, the more reliable sensor can then receive a higher vote. Oblivious/Adaptive input vote has to do with whether or not the votes are allowed to change at runtime(adaptive) or not (oblivious). Threshold/Plurality output vote has to do with whether or not the output vote has to be greater than some defined limit (threshold), or just an output with the highest support from the inputs(plurality).

It is clear that with the kind of sensor data provided by the ADCS the inexact input data class should be considered. For simplicity, the input votes will be oblivious and since we assume that every sensor is equally reliable this means that the vote for each sensor should be equal. The votes can then be ignored all together, this is a special case of the oblivious voting class [39].

Median, mean and plurality voting are some of the most common voting algorithms for the inexact and mediation class of voters [40].

8.1 Voting Algorithms

Mean Voter The arithmetic mean voter, or average voter, adds all the measurements and divides them with the number of measurements. The output data will be an average of all the input data.

Median Voter The median voter orders the input data and uses the median as the output data.

Plurality Voter The plurality voter compares the values for all of the input data and chooses the most common value as the output data. A certain limit on how similar the data have to be is required for the data to be defined equal. A plurality voter for inexact voting has been presented in [41].

Unfortunately we can not predict how a failing sensor will behave, the output might go to zero or some large or small value. The mean voter will give an error as soon as one of the sensors fails, especially if a failure means that the data goes to a really high or low value compared to the accurate data. The plurality voter should be able to handle one of the sensors failing, but if two sensors fails no clear majority will exists and the algorithm will likely fail. The median voter will also provide the correct output if one of the sensors fails, but it is harder to predict how it will behave if two sensors fails.

The median voter is simpler to implement than the plurality voter and more reliable than the mean voter in the case of the ADCS. Based on the short analysis of voting techniques presented here a median voter has been chosen and implemented in software. The exact same function can then be used for the gyroscope measurements and the magnetometer measurements. A simple insertion sort algorithm is being used to sort the arrays, this algorithm is efficient for small data sets and quite simple to implement. The algorithm is shown in Figure 8.2. Tests should be performed on hardware to see how large the difference between the three sensors are to better analyze how good the median voting technique will work. Performing tests in Matlab are less interesting, since the outcome is easily seen by

```
1   int median_voter(int16_t sensor_data[12]) {
2
3       int16_t temp_array[3] = {0, 0, 0};
4       int16_t temp = 0;
5
6       int j = 0;
7       int i = 0;
8       int k = 0;
9
10      for(k=0; k<3; k++) {
11          for(i=0; i<3; i++) {
12              temp_array[i] = sensor_data[(i+1)*3 + k];
13          }
14
15          for(i=0; i<3; i++) {
16              temp = temp_array[i];
17              j = i;
18              while (j > 0 && temp_array[j-1] > temp) {
19                  temp_array[j] = temp_array[j-1];
20                  j = j-1;
21                  }
22              temp_array[j] = temp;
23          }
24          sensor_data[k] = temp_array[1];
25      }
26
27  return 42;
28  }
```

Figure 8.2: Median Voter Algorithm

looking at the input data.

Chapter 9

Failure Analysis

In this chapter a short failure analysis for the ADCS will be performed to show how the ADCS might fail. Handling of failures have not been implemented in software, this chapter is meant as a foundation for any future work that might be performed on error handling. The analysis will show where errors might occur that can lead to the ADCS not operating as expected and a some methods to reduce the risk of errors are introduced. The various sources for failures can be seen in Figure 9.1.

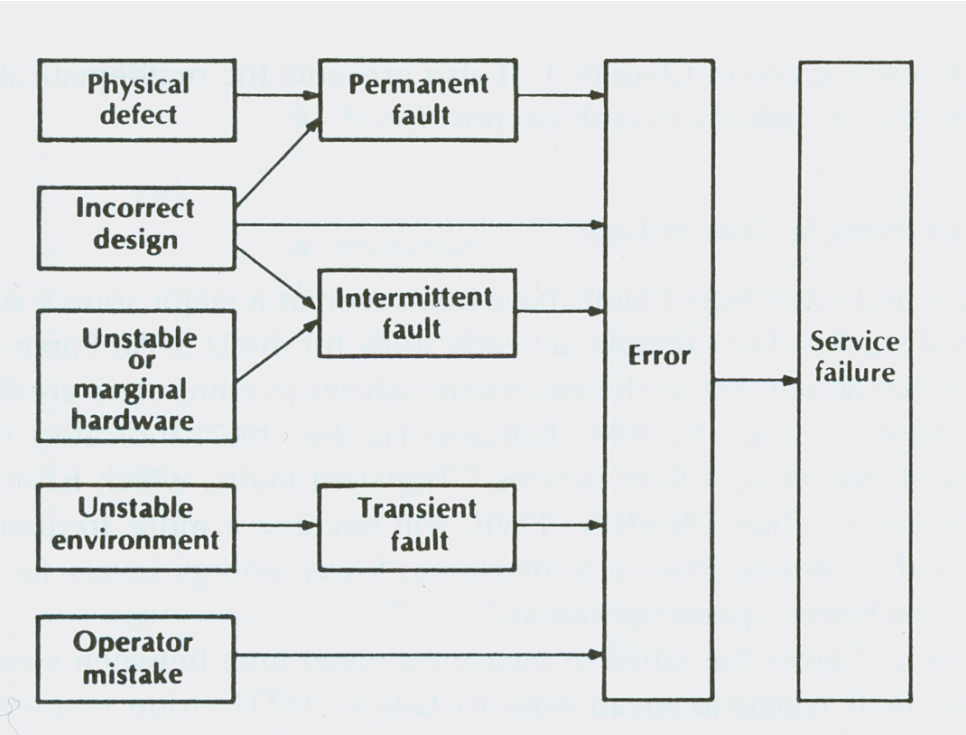

Figure 9.1: Error sources and their connections to service failures, figure taken from [17]

A service failure in the case of the ADCS means that it would no longer be able to perform

one, or several, of the operations it has been designed to perform: pointing, detumbling, estimation and tumbling (tumbling driven by the tumbling controller).

Operator mistakes are not a large issue for the ADCS since little to no operator input is required, the operator mistakes that can be made stems from uploading erroneous data to the ADCS. The environment will be unstable due to SEP and the highly varying temperatures, which could introduce a lot of errors. The hardware itself should not be unstable, but can become unstable du to SEP, total radiation effects and the highly varying temperatures. Incorrect design should be avoided by following the system definitions and coding standards vigorously and doing exhaustive testing on the system before launch. Physical defects might occur due to SEP, the total radiation dosage and temperature effects. Fault trees will now be used to show how the different operating modes might fail.

9.1 Fault Tree Analysis

A color coding scheme has been used for the fault tree analysis to simplify it. Green fields are software dependent, blue fields are hardware dependent, yellow fields can be caused by faults in both hardware and software, while red fields are the top level failures according to which services the ADCS should be able to provide. An estimation failure can be a top level failure, in the estimation mode, or it could be part of the chain causing a pointing failure in the pointing mode.

Figure 9.2 shows how the sensors and their respective processing algorithms can end up providing incorrect data to the different sensor data storage vectors. This fault tree is the same for all of the sensors because of the way the software and hardware is designed. The hardware components and software parts that can fail are off course different.

Figure 9.3 shows how the estimator can fail. The estimator is dependent on a lot of hardware, including all the sensors and the real time clock. This makes it very sensitive to errors in the hardware. A fault with the prediction algorithm also includes an error in the two line elements set, while a fault in the estimation algorithm also includes bad values for the Q and R matrices.

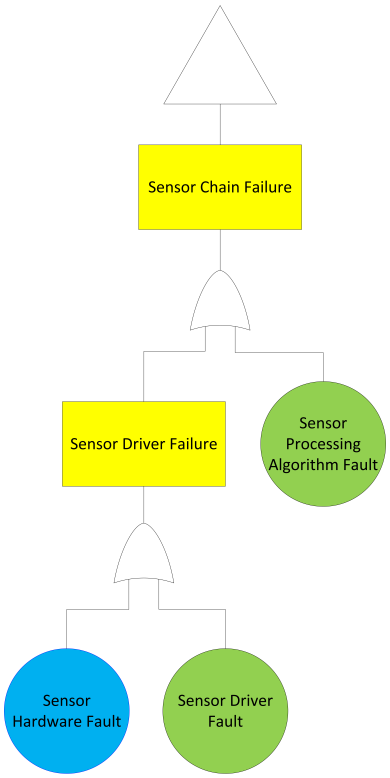

Figure 9.2: Fault tree for the sensors

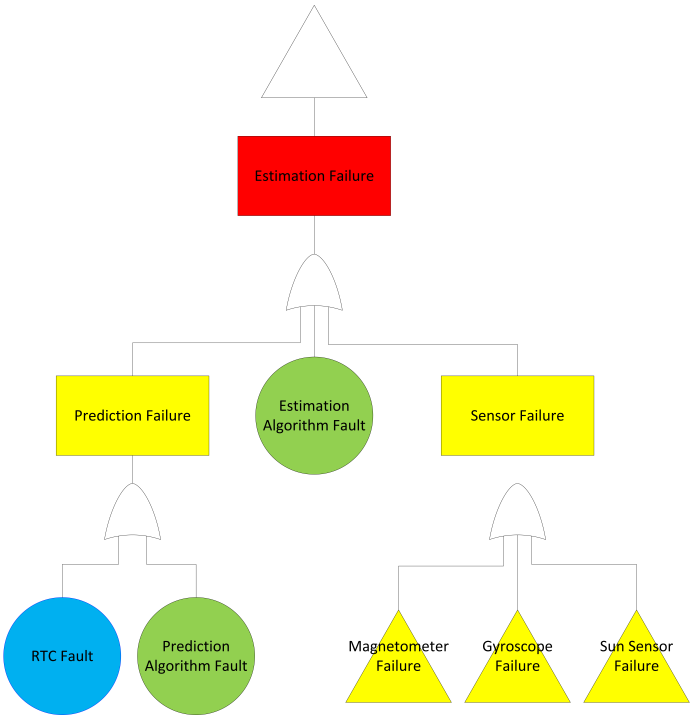

Figure 9.3: Fault tree for the estimation mode

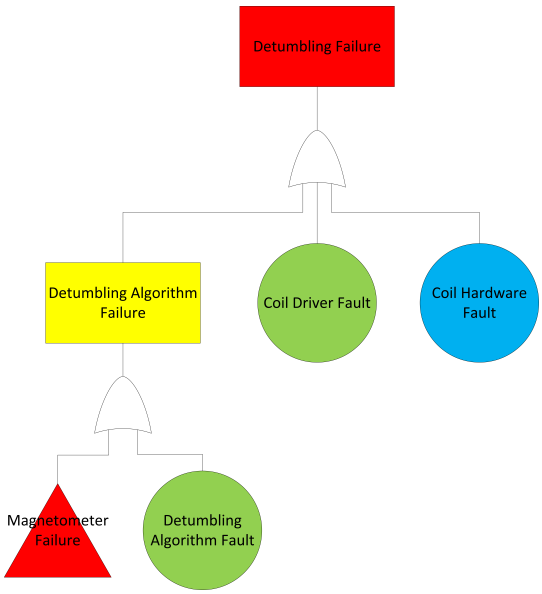

Figure 9.4: Fault tree for the detumbling mode

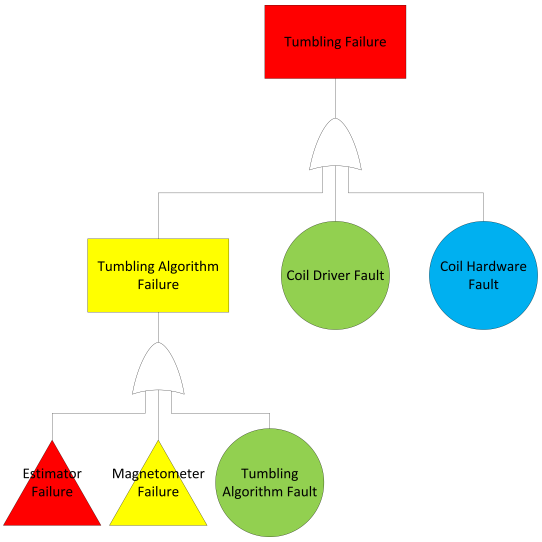

Figure 9.5: Fault tree for the tumbling mode

Figure 9.4 shows how the detumbling mode can fail. All of the modes that includes using any of the controllers will fail if there is a fault in the coil driver or coil hardware.

Figure 9.6 and Figure 9.5 shows how the pointing mode and the tumbling mode can fail. These are the longest chains, and thus have the highest probability of failing. They are dependent on the estimator, which means that they are also dependent on most of the hardware.

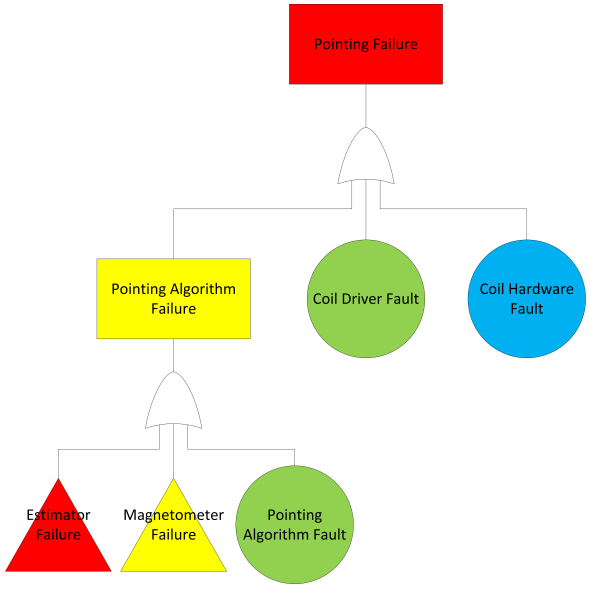

Figure 9.6: Fault tree for the pointing mode

It is important to note that in all of these fault trees a microcontroller fault (like a bit flip or any SEP) can occur at any point and cause a failure. This is not explicitly stated in the diagrams for simplicity. The failures can all be both transient, intermittent or permanent.

9.2 Failure Prevention

Transient faults leading to a wrong control output is not necessarily critical. If the control output is wrong during one cycle the controllers will still be able to perform their intended tasks, even tough it may take more time. Intermittent and permanent faults are of a larger concern, since using the wrong control output over a longer period of time might spin the satellite up to velocities larger than what the detumbling controller can tolerate. As can be seen from Figure 9.1 these errors comes from physical defects, incorrect design and unstable or marginal hardware. Any error handling implemented in software should therefore be focused on catching faults stemming from these sources.

Hardware

Hardware faults for the gyroscope and magnetometer measurements have been largely mitigated by having three sensors of each kind. The coils on the other hand have no redundancy

increasing the likelyhood of the coils failing completely. The hardware for each coil is independent of the other coils, but a failure in one coil is enough to make the control algorithms unstable. A failure of the RTC will lead to a timing error, that will most likely lead to a failure in the prediction algorithm. There exists an internal RTC on the microcontroller that can be used if the external RTC fails, an algorithm to detect an error on the external RTC and switch to the internal one could be developed. The sun sensors have some built in redundancy in the fact that the satellite will be equipped with more sensors than needed. Thus the algorithm to calculate the sun vector should be able to handle the loss of some of these sensors.

No more changes is intended to be made to the hardware, so errors coming from the hardware needs to be handled in software. This is of course not possible for malfunctioning components, like a H-bridge for one of the coils malfunctioning. This means that permanent faults in the coil driver components will lead to many of the ADCS modes not working. Permanent faults in more than one of the gyroscopes or magnetometers could also lead to a full failure of the operating modes, but this needs to be investigated more properly on the hardware. Permanent faults in two or more gyroscopes or magnetometers will lead to all of the operating modes failing, but it can be recovered by reprogramming the ADCS to only use any working sensors. All three of the sensors failing will off course lead to a failure of the ADCS and it will not be able to perform any of the tasks it was designed for.

Software

To avoid software faults there are several steps that can be taken for each part of the code. Having a highly modular system is in itself a good place to start to minimize software errors, as is having as little and as simple code as possible. All of the code should be exhaustively tested before launch. Watchdog timers should be implemented so that the system will be automatically reset if any parts of the code takes to long to finish, since this usually means that an error has occurred. Checksums and/or performing the algorithms twice to compare the results can be used to detect errors in the calculations and to see if any data have changed at any point it is not supposed to.

A program integrity check and a check of the flash memory could also be performed. More specifically a task can check the integrity of the variables it is using at startup. An error de-

tection and correction algorithm could then be developed to make sure that the code runs correctly. As already mentioned a specific task to deal with error correction and detection could be useful.

Chapter 10

Discussion and Results

10.1 Controllers and Simulations

Three controllers have been presented here, one for detumbling, one for pointing and one for tumbling. While the pointing and detumbling controllers had been chosen previously the tumbling controller were first presented with this purpose in this thesis. Several other controller alternatives exists for all modes, but the controllers presented here have been proven to work in several simulations. Stability evidence for the detumbling controller have been presented earlier in the NUTS project, and instability evidence for the tumbling controller have been presented here. Unfortunately no complete stability evidence for the pointing controller have been found, but a long range of simulations exists that indicates that it is stable.

The simulations presented here have shown that all of the controllers will work with the hardware and software design of the ADCS as it is now. That being said there have been found certain initial conditions from which the pointing controller have not been able to stabilize the satellite in the nadir pointing direction. If this is the case when the satellite is in space, the tumbling and detumbling controllers can be used to push the satellite into another attitude before retrying pointing.

The simulations of the pointing controller with noise is important because it sets a limit on how much noise it can be on the angular velocity estimate. The simulations shows that a certain amount of white noise is fine, but any slowly varying sinusoidal noise makes the

controller unstable. This is as expected, and the controller can be said to be robust. The estimator needs to produce estimates of the angular velocity that are as smooth as or smoother than what was found in this thesis. The combination of averaging the angular velocity data from the gyroscopes and running them trough the estimator should be enough to reduce the noise to tolerable levels.

To improve the performance of the pointing controller a more adaptive control scheme where the gains varied with the angular velocity was tried. Unfortunately this did not yield a better controller. That being said, there probably exists ways to tune the controller in this way to improve its performance, even though none have been found here. The controllers have simply been tuned to work as fast as possible with the actuator design. No more time should be used on looking into controllers for the NUTS project.

The tumbling controller has been implemented and tested in software. Testing has been performed in a black-box manner by comparing the output to the output of the corresponding Matlab code. For the previously chosen controllers the gains have been changed to the most recent findings, and the code has been updated to comply with the NUTS code standard.

10.2 Hardware

The hardware designed during this project has been designed in such a way that it should not be necessary with any more changes to the ADCS board. The only change that one might want to do is to add extra magnetometers somewhere else on the satellite, but the ADCS board itself has been designed to make this easy. The board should be flight ready with the current design. Unfortunately the hardware has not been tested, this means that there could be some errors.

The main difference between this version and the first one built is that there have been added more sensors for redundancy. There will not be any need to add more sensors, since having three of each is more then enough redundancy for a system such as this. A RTC has also been added to provide a more precise time solution, and preventing loss of time and date if a soft reset is needed for the ADCS. The new sensors have not been tested, but they should be good

enough for the ADCS.

The regular pins used on the first prototype to attach the sun sensors and coils have been replaced with lockable pins that should be able to resist vibrations during launch. The components have been placed on the ADCS board in a way that should minimize any electromagnetic noise and provide the most accurate data, but this has not been tested since the satellite is not yet built.

10.3 Software

As previously mentioned the ADCS is a collaborative project with several people involved both directly and indirectly. This has led to some software and the hardware not being finished, or finished late. Since one of the main goals of this thesis has been to design the software architecture for the ADCS this has had an effect on implementation and testing. Instead of the actual hardware a evaluation kit for the microcontroller has been used for development, and everything that can has been implemented and tested there. That being said the architecture design itself is completed and the framework has been implemented in software. Porting this implementation to the actual hardware should be a straightforward process.

The software has been designed according to the KISS (Keep it simple, stupid) principle. A description of the different subsystems used and how they should interact have been provided. Unfortunately, the ADCS group has not been following the NUTS code standard from the beginning, meaning that there are some large deviations in how code has been written. Any further work done on the software system should adhere strictly to the conventions in the NUTS code standard. The code as it is now is highly modular, and every subsystem can easily be replaced without having to change the overall system architecture. Having a highly modular system allows for easy black-box testing and makes error handling and correction easier.

All of the aspects discussed in Chapter 7 have been implemented and tested in software as best as possible without the hardware. Functions to read from and write the persistent

variables to the flash memory user page have been implemented and tested. These will have to be updated as the subsystems are finished. The logging system has been implemented and tested, but how often data should be logged and sent to the OBC have yet to be decided. The NRP subsystems have not been implemented or tested, but the code is finished, testing needs to be performed on the completed hardware.

Some of the subsystems, like the drivers for the hardware, have not been written. Also since the code is written by several different people and has arrived at different times, or not arrived at all, important tests like timing of the different subsystems have not been performed. A worst case assumption of 500 milliseconds to calculate a control output has been used in the system testing. The coil output has also been allowed to run for a maximum of 500 milliseconds. This means that the code will run periodically every second.

Two approaches for the software architecture have been presented since the system is not finished and proper testing have been impossible. Both approaches have worked on the evaluation kit of the microcontroller. Splitting the system into only two tasks is the simpler solution, and the program flow will then be controlled explicitly in a state machine in one of the tasks. Splitting the system into five tasks allows for a more flexible system, semaphores have been used to control the program flow. Semaphores will in both cases need to be used to control access to the global variables.

10.4 Sensor Redundancy Algorithm

A study has been performed on different algorithms to handle the sensor redundancy in software. Since we do not know how a failing sensor will behave predicting exactly what will happen with the different algorithms is impossible. Voting algorithms are the recommended algorithms for systems with hardware redundancy such as the ADCS. The gain in reliability one might get by using more advanced algorithms is offset by the increasing demands on runtime and the increased complexity.

A median voter has been chosen as the algorithm to process the sensor data for the gyroscopes and magnetometers to find a value close to the real value for the magnetic field vec-

tor and the angular velocity vector from the available measurements. This algorithm is fairly simple, but should be more than good enough for the ADCS. It has been implemented and tested in software and found to work as intended. Henrik Rudi Haave has been responsible for finding an algorithm to generate the sun vector from the available sun sensor data.

10.5 Failure Analysis

The failure analysis presented in this thesis is meant to show how the ADCS might fail to achieve the tasks it have been designed for. The pointing and tumbling modes have the highest probabilities of failure, since they are dependent on most code and hardware. The detumbling mode have a much smaller chance of failing. This is desirable since the detumbling mode is the most important mode, especially now that the satellite will not be equipped with a camera.

A more in depth Failure Mode and Effects Analysis (FMEA) could have been performed, but at this late stage that would not have been of much value since the amount of future changes in hardware and software should be limited to the minimum necessary to complete the system. It is still interesting to see how the system might fail, which has been shown here. The analysis presented here has not considered errors that can occur in other parts of the satellite that can cause an error in the ADCS, but it is important to note that this may happen. Failure handling has not been implemented in software.

Chapter 11

Conclusion

The project goals have been met. A tumbling controller has been found and shown to be unstable, while all of the controllers have been tuned to work well with the current actuator and software design of the ADCS. Simulations done in Matlab have been used for this purpose. Simulations have also been performed for the pointing controller to check robustness, and the controller have been found to be robust. The estimator has to be able to provide estimates of the angular velocity that are as smooth or smoother than what was found here.

A second version of the ADCS board has been designed. The hardware includes several sensors of each kind for redundancy as well as some other upgrades to make it more reliable. A study on some algorithms to handle the hardware redundancy in software has been performed, and a median voter algorithm has been implemented for the gyroscopes and magnetometers. No more changes should be necessary for the hardware or the control algorithms before launch.

The software architecture for the ADCS has been designed. Data storage variables and the different subsystems have been defined. The system is running on FreeRTOS and has been split into tasks, data sharing between the tasks are done by global variables. A set of commands that can be sent to the ADCS have also been defined. The system is highly modular, and an effort has been made to keep the code as simple as possible. Most of the subsystems have been implemented, and the task framework is in place. Unfortunately the hardware have not been tested, and the software has not been tested on the designed hardware, only a evaluation kit for the chosen microcontroller.

Simple code often leads to less errors, but the ADCS may still fail. To investigate how a short failure analysis was performed, fault trees have been used to show where errors for the different operating modes can originate. A short discussion on how the minimize the risk of errors have been done, software such as watchdog timers and some integrity checks should be implemented when the system is completed.

Chapter 12

Future Work

The future work needs to focus on finishing the software.

- Implement the driver subsystems.
- Update the estimator algorithm to include an estimate of the angular velocity.
- Implement an algorithm to create a sun vector from the available sun sensor measurements.
- Test the hardware.
- Implement and test NRP with the rest of the ADCS software.
- Finish the software system.
- Test the software system.

Some error handling could, and probably should, be implemented in software to make the system more reliable. After all of the items above have been performed the ADCS should be ready for launch.

Appendix A

Data Structures

This appendix states witch data structures exists in the system, the persistent variables we will keep and which subsystems who should be able to access what data. A more complete list can be found in the digital attachments.

Table A.1: Global variables used in ADCS to store data

Name	**Size**	**Data Type**
mode	1	int
gyroscope_data	12	int16_t
magnetometer_data	12	int16_t
sun_sensor_data	19	uint16_t
predictor_data	12	double
estimator_data	16	float
coil_output_data	3	int16_t
temperature_data	6	int16_t

The angular velocity vector found by the median voter algorithm should be stored first in the gyroscope data, then the gyroscope measurements. The same goes for the magnetometer data and sun sensor data. Data should be stored in the order: $[x, y, z]$, for all of the different components.

Table A.2: Persistent variables to be stored in the user page

Variable Name	**Data**	**Belongs to Subsystem**
mode	int	Global
TLE	char[136]	Predictor
R	float[9]	Estimator
Q	float[25]	Estimator

Table A.3: Global data structures used in the ADCS to store data

Structure Name	**Data**	**Data Type**
Time		
	year	int16_t
	mon	int16_t
	day	int16_t
	hr	int16_t
	min	int16_t
	sec	double
Data		
	mode	int
	time	Time
	gyroscope_data[12]	int16_t
	magnetometer_data[12]	int16_t
	sun_sensor_data[19]	uint16_t
	predictor_data[12]	double
	estimator_data[16]	float
	coil_output_data[3]	int16_t
	temperature_data[6]	int16_t

The data structure Data is used for logging and to have a defined structure to send all current data in. The mode variable needs to be readable from all tasks, it is not used in the subsystems directly.

Table A.4: Table showing which subsystem needs access to what global data

Subsystem	**Read**	**Write**
Gyroscope Driver	NONE	gyroscope_data temperature_data
Magnetometer Driver	NONE	magnetometer_data temperature_data
Sun Sensor Driver	NONE	sun_sensor_data
Coil Driver	coil_output_data	NONE
Gyroscope Data Processing	gyroscope_data	gyroscope_data
Magnetometer Data Processing	magnetometer_data	magnetometer_data
Sun Sensor Data Processing	sun_sensor_data	sun_sensor_data
Predictor	Time	predictor_data
Estimator	gyroscope_data, magnetometer_data, sun_sensor_data, predictor_data	estimator_data
Detumbling Controller	magnetometer_data	coil_output_data
Pointing Controller	magnetometer_data, estimator_data	coil_output_data
Tumbling Controller	magnetometer_data, estimator_data	
Logg	ALL	NONE
NRP	ALL	mode, Time

Appendix B

Messages

This appendix details a set of messages that have been identified as necesseary or good to have for the ADCS. More messages will probably be added before launch as the different subsystems is finished and new demands and ideas come up.

ADCS In

Table B.1: Messages that can be sent to the ADCS

Description	**Command**	**Data**
Change mode	SET_MODE	int mode
Get ADCS mode	GET_MODE	NONE
Get ADCS time	GET_TIME	NONE
Get ADCS log	GET_LOG	NONE
Get ADCS data	GET_DATA	NONE
Get ADCS temperatures	GET_TEMP	NONE
Update ADCS time	UPDATE_TIME	Time time
Update predictor TLE	UPDATE_PREDICTOR_TLE	char predictorTLE[139]
Update estimator Q matrix	UPDATE_ESTIMATOR_Q	int16_t Q[25]
Update estimator R matrix	UPDATE_ESTIMATOR_R	int16_t R[9]

ADCS Out

Table B.2: Messages that can be sent from the ADCS

Description	**Reply To**	**Data**
Send mode	GET_MODE	int mode
Send time	GET_TIME	Time time
Send log	GET_LOG	Data data[LOG_SIZE]
Send data	GET_DATA	Data data
Send temperatures	GET_TEMP	int16_t temperature_data[6]

Appendix C

Sequence Diagrams

Here the sequence diagrams for all the states are presented. To make the diagrams smaller the magnetometer driver, gyroscope driver and sun sensor driver have been moved into one block called "Sensors". For the same purpose the functions to process the sensor data for the magnetometer, gyroscope and sun sensor data have been moved into one block called "Sensor Processing". Writing data to the log should be performed at the end of each diagram, this is not shown explicitly.

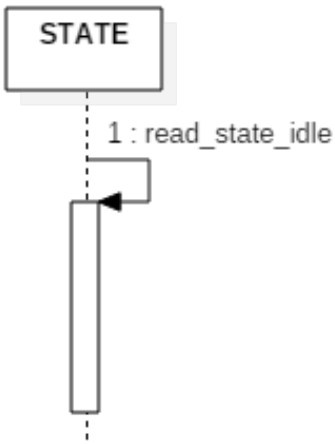

Figure C.1: Idle state sequence diagram

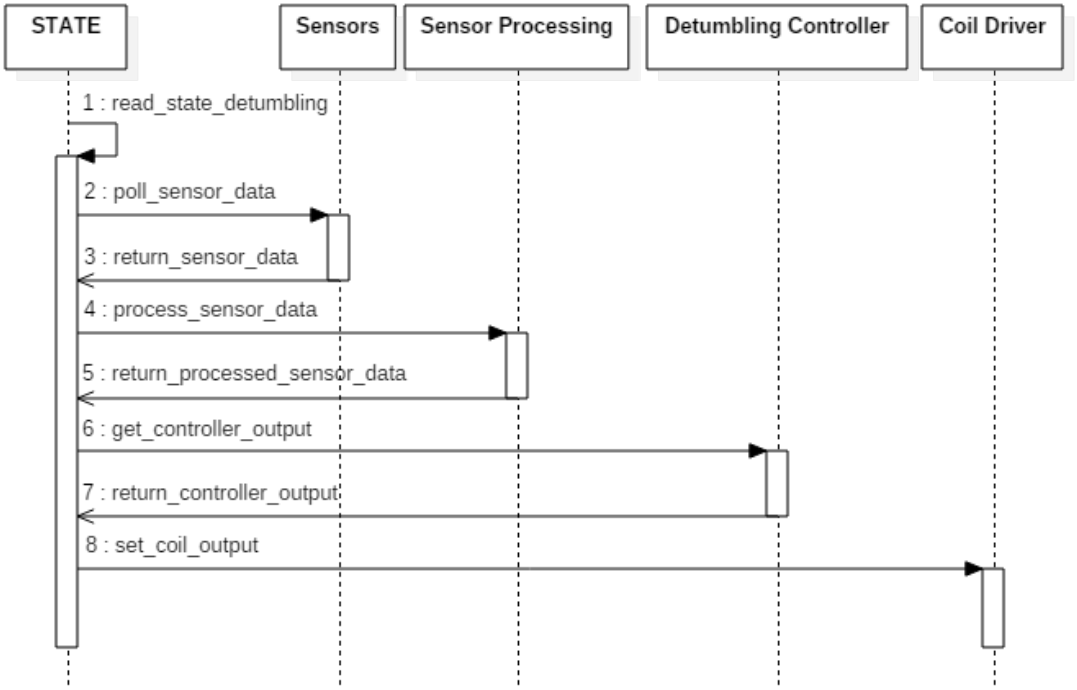

Figure C.2: Detumbling state sequence diagram

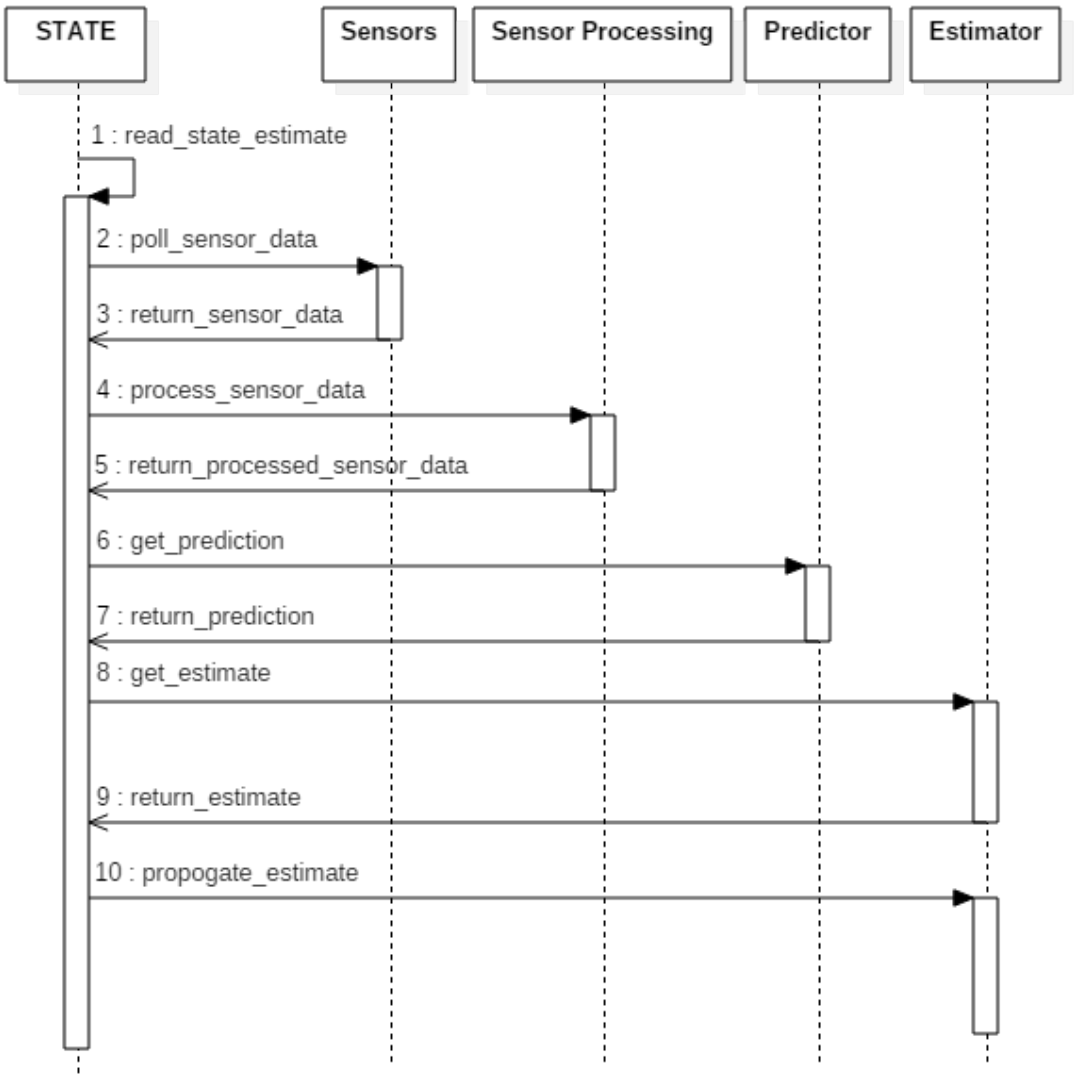

Figure C.3: Estimate state sequence diagram

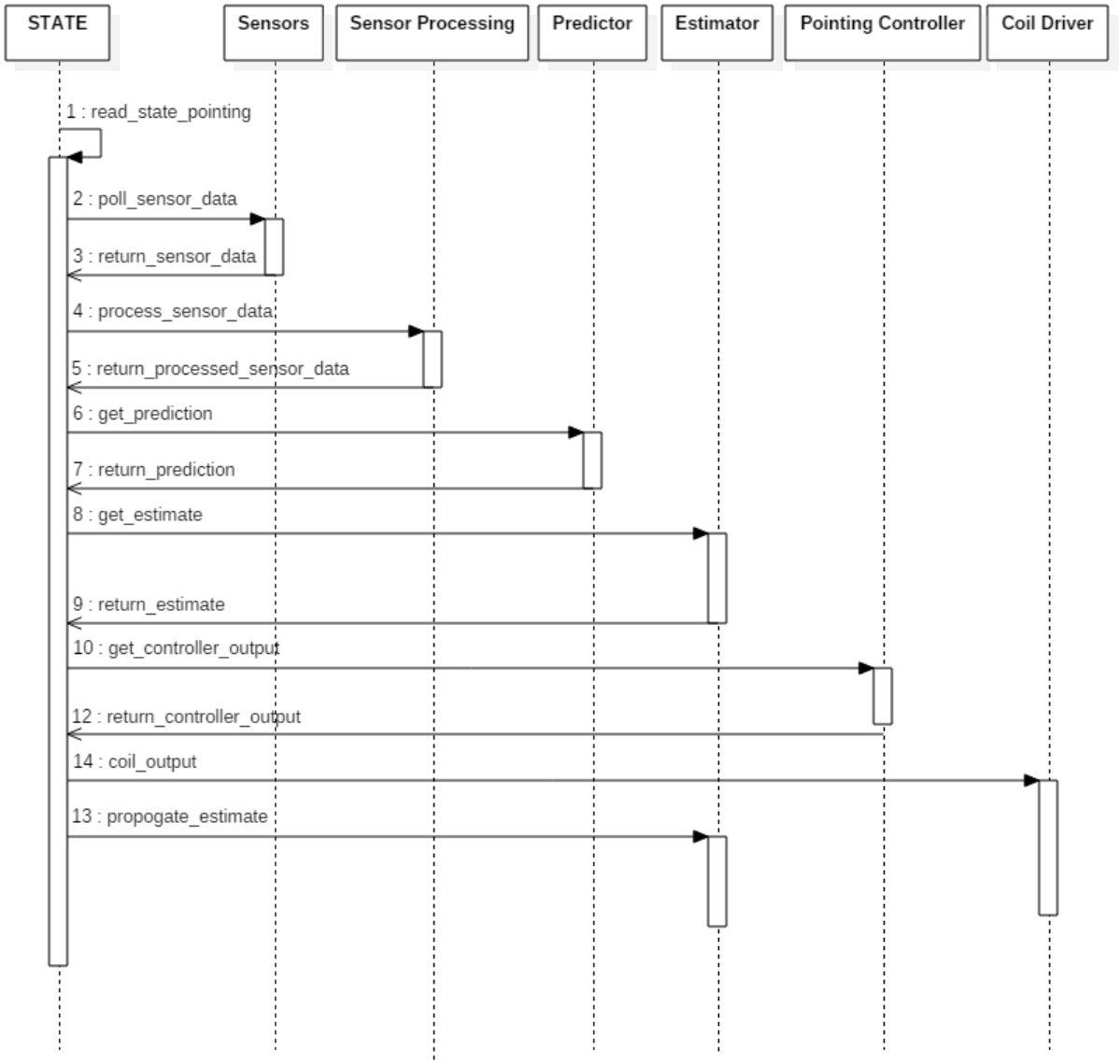

Figure C.4: Pointing state sequence diagram

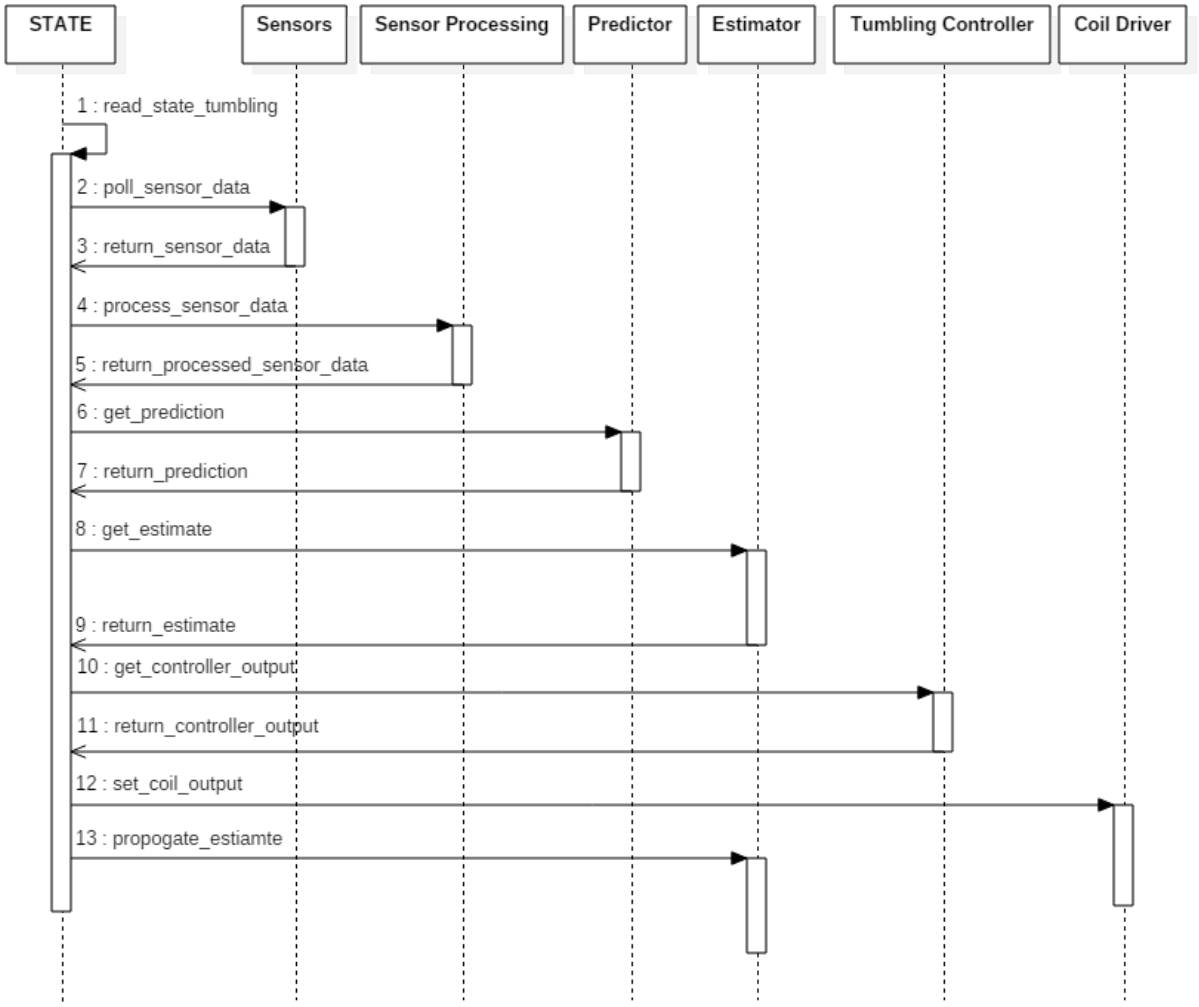

Figure C.5: Tumbling state sequence diagram

Bibliography

[1] The CubeSat Program, Cal Poly SLO. Cubesat Design Specification. http://www.cubesat.org/images/developers/cds_rev13_final2.pdf. Accessed: 2015-06-16.

[2] Gaute Bråthen. *Design of Attitude Control System of a Double CubeSat*. Master thesis, NTNU, 2013.

[3] Zdenko Tudor. *Design and Implementation of Attitude Control for 3-axes Magnetic Coil Stabilization of a Spacecraft*. Master thesis, NTNU, 2011.

[4] Fredrik Alvenes. *Attitude Controller-Observer Design for the NTNU Test Satellite*. Master thesis, NTNU, 2013.

[5] Marius Fløttum Westgaard. *Hardware Testing of Attitude Control for the NTNU Test Satellite*. Project thesis, NTNU, 2014.

[6] Per Kolbjørn Soglo. *3-Aksestyring av Gravitasjonsstabilisert Satelitt ved bruk av Magnetspoler*. Master thesis, NTH, 1994.

[7] Kristian Lindgård Jenssen, Kaan Huseby Yabar, and Jan Tommy Gravdahl. A comparison of attitude determination methods: theory and experiments. In *Proceedings of the 62nd International Astronautical Congress*, October 2011.

[8] Toril Bye Rinnan. *Development and Comparison of Estimation Methods for Attitude Determination*. Master thesis, NTNU, 2012.

[9] Henrik Rudi Haave. *Implementation of Attitude Estimation and a Look at the UWE CubeSat*. Project thesis, NTNU, 2014.

[10] Øyvind Rein. *Developing an ADCS Prototype for NTNU Test Satellite*. Master thesis, NTNU, 2014.

[11] Antoine François Xavier Pignède. *Prediction Algorithms for the NUTS Attitude Estimator and Robust Spacecraft Attitude Stabilization using Magnetorquers.* Master thesis, NTNU, 2015.

[12] Jan Tommy Gravdahl and Olav Egeland. *Modeling and Simulation for Automatic Control.* Marine Cybernetics, 2002.

[13] Thor I. Fossen. *Handbook of Marine Craft Hydrodynamics and Motion Control.* Wiley, 2011.

[14] Peter C. Hughes. *Spacecraft Attitude Dynamics.* John Wiley & Sons, 1986.

[15] Kjell Arne Ødegaard. *Error Detection and Correction for Low-Cost Nano Satellites.* Master thesis, NTNU, 2013.

[16] NASA. Space Radiation Effects On Electronic Components In Low-Earth Orbit. http://engineer.jpl.nasa.gov/practices/1258.pdf. Accessed: 2015-05-20.

[17] Daniel P. Siewiorek and Robert S. Swarz. *Reliable Computer Systems: Design and Evaluation.* Digital Press, 2nd edition, 1992.

[18] Tor Onshus. *Instrumenteringssystemer.* Department of Engineering Cybernetics, NTNU, 5th edition, 2011.

[19] Antoine François Xavier Pignède. *Detumbling of the NTNU Test Satellite.* Project thesis, NTNU, 2014.

[20] Torben Graversen, Michael Kvist Frederiksen, and Søren Vejlgaard Vedstesen. *Attitude Control system for AAU CubeSat.* Masther thesis, Aalborg University, 2002.

[21] A. Craig Stickler and K. T. Alfriend. Elementary Magnetic Attitude Control System. *Journal of Spacecraft and Rockets,* Vol. 13(No. 5), 1976.

[22] Rafal Wisniewski. *Satellite Attitude Control Using Only Electromagnetic Actuation.* Ph.d thesis, Aalborg University, 1996.

[23] Hassan K. Khalil. *Nonlinear Systems, Third Edition.* Prentice Hall, 2002.

[24] M. Vidyasagar. *Nonlinear Systems Analysis.* Prentice-Hall International, 2 edition, 1993.

[25] Wikipedia. Pulse Width Modulation. https://en.wikipedia.org/?title=Pulse-width_modulation. Accessed: 2015-06-16.

[26] MSS. Marine Systems Simulator. http://www.marinecontrol.org. Accessed: 2015-02-10.

[27] SwissCube. http://swisscube.epfl.ch. Accessed: 2015-03-04.

[28] Honeywell. HMC5983 Datasheet. http://www.farnell.com/datasheets/1802211.pdf. Accessed: 2015-04-16.

[29] Maxim Integrated. MAX21000 Datasheet. http://datasheets.maximintegrated.com/en/ds/MAX21000.pdf. Accessed: 2015-04-16.

[30] STMicroelectronics. L3G4200D Datasheet. http://www.st.com/web/en/resource/technical/document/datasheet/CD00265057.pdf. Accessed: 2015-04-16.

[31] Atmel. AT32 UC3C Series Datasheet. http://www.atmel.com/Images/doc32117.pdf. Accessed: 2015-02-25.

[32] AVX. KT3225T32768EAW30TAA Datasheet. http://www.mouser.com/ds/2/40/kr3225y_e-514863.pdf. Accessed: 2015-04-16.

[33] Molex. Pico-EZmate 078171-5006 Datasheet. http://www.molex.com/pdm_docs/ps/PS-78172-001.pdf. Accessed: 2015-04-16.

[34] AAUSAT3 CubeSat. https://stemn.com/projects/aausat3-cubesat. Accessed: 2015-03-14.

[35] FreeRTOS. Free Real Time Operating System. http://www.freertos.org. Accessed: 2015-03-04.

[36] CSP. Cubesat Space Protocol. https://github.com/GomSpace/libcsp. Accessed: 2015-03-04.

[37] Erlend Riis Jahren. *Design and Implementation of a Reliable Transport Layer Protocol for NUTS.* Master thesis, NTNU, 2015.

[38] Alan S. Willsky. A survey of Design Methods for Failure Detection in Dynamics Systems. *Automatica,* Vol. 12:pp. 601–611, 1976.

[39] Behrooz Parhami. A taxonomy of voting schemes for data fusion and dependable computation. *Reliability Engineering and System Safety*, Vol .52:pp. 139 –151, 1996.

[40] Behrooz Parhami. Voting algorithms. *IEEE Transactions on Reliability*, Vol. 43(No. 4), 1994.

[41] D.M. Blough and G.F. Sullivan. A comparison of voting strategies for fault-tolerant distributed systems. *Proceedings Ninth Symposium on Reliable Distributed Systems*, 1990.