NTNU

Norwegian University of
Science and Technology

Design and Implementation of Attitude Control for 3-axes Magnetic Coil Stabilization of a Spacecraft

Zdenko Tudor

Master of Science in Engineering Cybernetics
Submission date: May 2011
Supervisor: Jan Tommy Gravdahl, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

THESIS

DESIGN AND IMPLEMENTATION OF ATTITUDE CONTROL FOR 3-AXES MAGNETIC COIL STABILIZATION OF A SPACECRAFT

Submitted by

ZDENKO TUDOR

Institute of

Engineering Cybernetics

Submitted in Partial Fulfillment of the Requirements For the Degree of

MASTER OF SCIENCE

Norwegian University of Science and Technology

Spring 2011

Master's Committee:
Advisor: Jan Tommy Gravdahl

Roger Birkeland

Abstract. Spacecrafts, especially satellites, play an ever greater role in our daily lives as we increasingly depend on the services they provide, which in turn, more often than not, critically depend on maintaining correct payload attitude. As smaller educational satellites pave the way for organization, group and privately owned pico-satellites, we explore the possibilities of attitude control through magnetic coil actuation. We approach the whole problem, from control theory development to first prototype actualization and control algorithm implementation, presenting the steps taken in a user-friendly manner while pointing out the pitfalls and drawbacks of different solutions. The control is based on a dissipative detumbling controller which after the initial phase is overridden by the reference controller attaining final desired payload attitude. We find that a simple 8-bit, 16Mhz microcontroller unit has the sufficient processing power to continuously compute the geomagnetic field using the complex International Geomagnetic Reference Field model, while simultaneously maintaining correct coil actuation. The power consumed by the controllers during the < 300 minute control phase, from initial tumbling to desired attitude, given a typical tumbling velocity of absolute magnitude $0.2\frac{rad}{s}$, is found to be no more than 150 Joules across the randomly selected test scenarios. Thus we are able conclude that three perpendicular magnetic coils, together with constantly present disturbances and complex geomagnetic field model preventing it from remaining at an ill-aligned attitude where one actuating degree of freedom is lost, provide sufficient actuation for reference control of a spacecraft.

Preface

In 2006, the Norwegian Center for Space-related Education (NAROM), the Norwegian Space Center (NSC) and Andøya Rocket Range (ARR) decided to initiate the Norwegian Student Satellite Program (ANSAT). This project was to be a continuation of the nCube-project and its scope; to build and launch 3 CubeSats in the period 2007 – 2014.

The three satellites are planned to follow the CubeSat standard. The standard describes the satellites as put together of cubes weighing no more than 1.33kg and measuring 10x10x10cm. The satellites can be single, double or triple cubes, all connected along the same axis. The first two of the three satellites are the HINCUBE, being developed by the students from Narvik University College and CUBESTAR, developed by students at the University of Oslo.

In September 2010, the Norwegian University of Science and Technology (NTNU) signed an agreement with NAROM to develop and build the last of the three CubeSats named NTNU Test Satellite (NUTS). This was to be a two unit CubeSat - a $10 \times 10 \times 20$cm satellite.

A Master's thesis proposal regarding attitude control of the NUTS satellite was among others issued by Jan Tommy Gravdahl at the Institute for Engineering Cybernetics, NTNU. Attitude control, together with attitude determination, combine into the Attitude Determination and Control System (ADCS), which is required to make use of the intended scientific camera payload. This thesis is thus written as a start phase for the first prototype of the satellite's Attitude Control module.

Before moving on, we wish to thank our supervisor, Jan Tommy Gravdahl

for being one of the most down-to-earth professors a student can encounter, and for being so easy to relate to, thus keeping the motivation high.

We also wish to thank all the peer students working on the satellite: Kaan Huseby Yabar and Kristian Lindgård Jenssen for constantly being there to bounce ideas off of and for co-operation throughout the whole project; Dewald De Bruyn and Marius Volstad for their assistance with practical electronics components and issues; Fredrik Sola Holberg for his help with the ODE output functions, and Roger Birkeland for keeping us all productive and helping out in every way possible.

We would also like to thank John Olav Horrigmo at the Engineering Cybernetics institute for helping out with the practicalities when making the prototype PCB; Oddvar Landrø at the Department of Electric Power Engineering for spinning the prototype coil; Johannes Skaar for providing insight on magnetic coils; Terje Mathiesen for helping with the temperature test chamber and finally we wish to thank the institute of engineering cybernetics, NAROM and all others providing the funding and support for such a innovative project - thus making it a reality.

To my young brother Isak; may he grow up to explore the stars.

NTNU
Norges teknisk-naturvitenskapelige universitet

Fakultet for informasjonsteknologi, matematikk og elektroteknikk
Institutt for teknisk kybernetikk

MSc thesis assignment

Name of the candidate: Zdenko Tudor

Subject: Engineering Cybernetics

Title: Attitude control for the Norwegian student satellite NUTS

Background

The topic of the assignment is attitude control of satellites using magnetic actuation. The assignment is part of the NUTS-project.

Assignment:

1) Present previous work on the following topics:
    a. Dynamic models describing the dynamics and kinematics of a magnetic actuated satellite.
    b. Nonlinear controllers for attitude of a magnetic actuated satellite.
2) Implement a simulator in Simulink for the satellite. Also include a model of the geomagnetic field, using a) the dipole model and b) the IGRF-model.
3) Investigate the power consumption of the control system and evaluate this versus the proposed available power of the satellite.
4) Design a prototype of the control system for one actuator.

To be handed in by:

Trondheim,

Jan Tommy Gravdahl

Jan Tommy Gravdahl
Professor, supervisor

Contents

Nomenclature

δ	Angular deviation from polar orbit
ϵ	Euler axis
η	Euler angle
λ	Latitude
ω_{IB}^{B}	Angular velocity of the Body frame with respect to the Inertial frame, expressed in Body frame
ω_{OB}^{B}	Angular velocity of the Body frame with respect to the Orbit frame, expressed in Body frame
ω_o	Rotation speed of a satellite constantly pointing towards the earth
θ	Longitude
$A_\circ$	Areas of the $\circ$ face coil
$c_\circ$	$\circ$-th column of the $\mathbf{R}_O^B$ matrix
$i_\circ$	Current running through the $\circ$ face coil
$N_\circ$	Number of turns in the $\circ$ face coil
$R_\circ$	Resistances in the $\circ$ face coil
V_{BB}	Load supply voltage
V_{CC}	Logic supply voltage
x_b, y_b, z_b	Body frame unit axes vectors

x_e, y_e, z_e	Earth Centered Earth Fixed (ECEF) frame unit axes vectors
x_i, y_i, z_i	Earth Centered Inertial (ECI) frame unit axes vectors
x_n, y_n, z_n	North East Down (NED) frame unit axes vectors
x_o, y_o, z_o	Orbit frame unit axes vectors

List of Figures

List of Tables

Lists

Acronyms and Abbreviations

ADCS	Attitude determination and Control System
IGRF-11	International Geomagnetic Reference Field, 11th generation
SV	Secular Variation
PWM	Pulse Width Modulation
ECI	Earth Centered Inertial frame
ECEF	Earth Centered Earth Fixed frame
NED	North East Down frame
AWG	American Wire Gauge
NAROM	Norwegian Center for Space-related Education
IAGA	International Association of Geomagnetism and Aeronomy
NASA	National Aeronautics and Space Administration
IERS	International Earth Rotation and Reference Systems Service
NOAA	National Oceanic and Atmospheric Administration
NGDC	National Geophysical Data Center

Instruments

PCB plotter	LPKF ProtoMat S62
Oscilloscope	Agilent Technomogies DSO3152A Digital Oscilloscope
Separating transformer	Generic (no instrument tag)
Multimeter	Fluke 289 True RMS Multimeter
Signal generator	ISO-TECH GFG8219
Temperature test chamber	Vötsch Industrietechnik VT 4011

Chapter 1

Introduction

Creating a non-commercial satellite, able to operate in extreme conditions of outer space, using only commercial off-the-shelf components is a daunting task to say the least. This is however a requirement when operating with a marginal budget.

Nevertheless, the NUTS CubeSat is to be a fully functional $10 \times 10 \times 20$cm satellite with the payload being a nightvision camera. With this camera the final mission objective, beyond its immense educational value, is to acquire images of atmospheric waves as they pass through an airglow layer. The reason we wish to study these is that they are the primary way in which atmosphere exchanges energy up and down between the climate and weather region (0-12km) and the upper atmosphere (90km). As such, they play a major role in the climate and weather of the earth, but their global and seasonal variations are not yet quantified.

The camera mounted on the satellite needs to stand rather still and point towards the Earth in order to gather any useful imagery of this phenomenon. This is where attitude control, which is the topic of this thesis, is utilized.

In this paper we discuss the whole procedure of designing a working attitude control module; from theoretical analysis and simulations, through hardware choice, programming and design; finally ending in hardware testing.

In our experience, the transitional costs of moving the project onward to a suceeding group are quite large. Thus we try to be as detailed as possible when presenting our results, providing insights and shedding light on the possible pitfalls when designing a system of this sort. In order to do this we include a great number of figures to help explain concepts at hand. We also attach Matlab and C code in the appendix to make things as clear as possible. With this we wish to plant the seed for a successful development and launch of the NUTS satellite in the years to come.

1.1 Previous Work

Early work consists mostly of the research done for the nCube2 and nCube1 projects which were launched on 27.10.05 and 26.07.06 respectively. Unluckily the nCube2 team was never able to make radio contact with the satellite, while nCube1 was destroyed during the second launch phase, when an error was detected with the carrier rocket. There are thus no indicators, even after two unsuccessful missions, that there is anything wrong with the existing control theory which has been rigorously simulated if not analytically proven functional. We therefore focus on re-implementing and improving the existing control algorithms.

The best of the earlier works, the major leap in our opinion, was done by Per Kolbjørn Soglo in his MSc thesis [20]. Soglo develops all the necessary control algorithms, using the theory from Spacecraft Attitude Dynamics by Peter C. Hughes [7]. In [12] Raymond Kristiansen creates the Matlab code for the rather analytically complex Earth's magnetic field model (IGRF11). Other noteworthy work is done by e.g. Fauske and Makovec in [5, 14] respectively. Some of these, but also many other can be found on Jan Tommy Gravdahl's web page [9].

1.2 Thesis Outline

Chapter 2: Introduces the mathematical preliminaries for the thesis, involving notations and definitions. This chapter also includes some Lyapunov stability theorems.

Chapter 3: Explains and models the dynamics of a rigid body in orbit - our satellite. Derives equations for gravity gradient torque and satellite's potential energy. Introduces the dipole and IGRF11 geomagnetic models.

Chapter 4: Presents the two controllers to be implemented: a detumbling controller and a reference controller for large deviations. We provide the reasoning and theory behind these two controllers but omit stability analysis, which has already been done in e.g. [20]. The principle of pulse-width-modulation is explained as well.

Chapter 5: Compares the IGRF and dipole models. Shows results of the satellite simulations done in Matlab, and attempts to provide some basic intuition behind controller's behavior.

Chapter 6: Explains the reasoning and approach behind hardware choice and design. This chapter is particularly important for hardware novices as it points out possible pitfalls. We also provide reasoning for pulse-width-modulation initialization in the C-code.

Chapter 7: Presents the results of the tests done on the final prototype. Points out which aspects need further analysis in the future.

Appendix A: Provides additional simulation figures which may help in understanding the simulations perormed in Chapter 5.

Appendix B: Here we include an excerpt from the American Wire Gauge, the IGRF-11 coefficient table and an illustration of the IGRF model.

Appendix C: All the essential programming code is included here. It is primarily intended for those attempting to recreate the results in this thesis.

Chapter 2

Background and Mathematical Notation

We begin with the mathematical background and notations necessary to grasp the concepts and ideas presented in later chapters. More specifically, we discuss the reference frames needed for our purposes, matrix and vector notations and rotation matrices together with their quaternion representations. Most of these concepts have carefully been described earlier in similar theses such as [20, 5, 12] by Soglo, Fauske and Kristiansen respectively. Due to the innate nature of this thesis, the mathematical background presented follows that of the earlier work closely. In addition most of the mathematical notation expressed here is in line with that in the Modelling and Simulations book by Olav Egeland and Jan Tommy Gravdahl [4], with slight additions from [7] to hopefully avoid any possible confusion.

2.1 Reference Frames

The most convenient set of reference vectors used to describe both the position and orientation of a satellite, is a *dextral orthonormal triad*[1] as described by Peter C. Hughes in [7]. Such a triad is given the descriptive term *reference frame.* There is a number of coordinate reference frames used when describing a satellite's position and its attitude. The particular reference frame used is a function of what one is attempting to achieve or describe. The frames

[1]Dextral - right handed. Orthonormal - Mutually perpendicular and of unit length. Triad - group of tree, a triplet

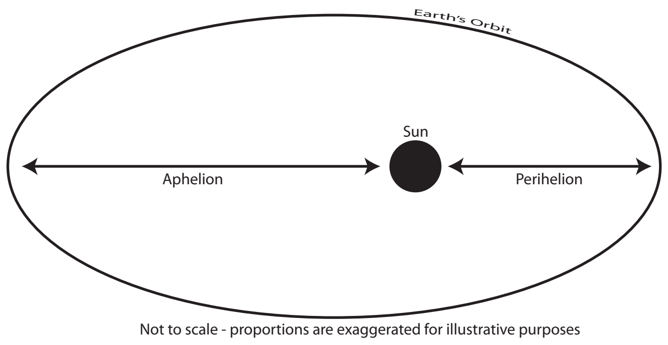

Earth's Orbit
Sun
Aphelion
Perihelion
Not to scale - proportions are exaggerated for illustrative purposes

Figure 2.1: Aphelion and Perihelion.

used in this thesis are presented below.

2.1.1 Earth Centered Inertial (ECI) Frame

Earth Centered Inertial frames represent a group of coordinate frames with their origins at the center of mass of the Earth defined by its unit axis vectors: x_i, y_i, and z_i. For our purposes we will define this frame in the following manner: z_i-axis is directed towards the celestial north pole, along the Earth's rotation axis, x_i-axis points towards the northern hemisphere's *vernal equinox*[2]. The y_i axis is defined by the right hand rule to complete a right hand orthogonal frame. From this we see that the fundamental plane for the ECI defined here is the Earth's equatorial plane. The ECI frames are not truly inertial since the Earth itself accelerates as it travels in its orbit around the Sun, due to its elliptical orbit. The eccentricity of the orbit can be found using: $\epsilon = \frac{a-p}{a+p}$ where a is the Earth's distance to the Sun at *aphelion* and p the distance at *perihelion* as illustrated in Figure (2.1). From the Science NASA webpage, [15], we obtain $a = 152.1 \cdot 10^6$m and $p = 147.5 \cdot 10^6$m which give us the eccentricity $\epsilon \approx 1.67 \cdot 10^{-2}$. Such low eccentricity for our purposes can be neglected and we consider this frame as inertial.

[2]The point where the sun crosses the Earth's equatorial plane, going from south to north. The x_i axis is parallel to the line from the center of the Earth to the Sun on the first day of spring in the northern hemisphere, which happens approximately on March 21. On this day, at the equator, the sun is directly overhead. For further illustration the reader is encouraged to see Figure 3.4 in [14].

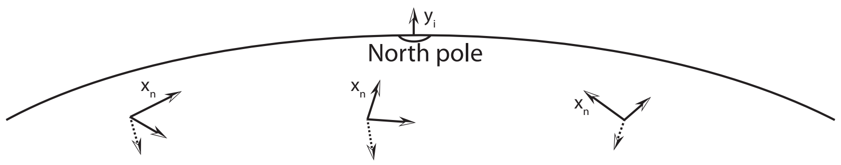


Figure 2.2: NED frames for different longitudes.

2.1.2 Earth Centered Earth Fixed (ECEF) Frame

The Earth Centered Earth Fixed frame differs from the ECI frame by the fact that it rotates together with the Earth. As the name suggests, the ECEF has its origin at the center of mass of the Earth. We define the orientation of the frame by the following unit axis vectors: x_e, y_e and z_e. As in the ECI frame the z_e-axis is directed along the Earth's rotational axis towards the celestial north pole, the x_e-axis is directed along the intersection of 0° latitude and 0° longitude half-planes while the y_e-axis is again defined such that x_e, y_e and z_e span a right hand frame vector space. The constant angular velocity of the ECEF frame is easily found as the $\frac{2\cdot\pi}{p_e}$, where p_e represents the Earth's rotation period (one day) in seconds. The rotation period relative to fixed stars, called a *stellar day* is $p_e \approx 86164$s which gives us the ECEF rotation $\omega_e = 7.2921 \cdot 10^{-5}$. This constant is confirmed on p. 19 in [8].

2.1.3 North East Down (NED) Frame

The North East Down frame is mounted on the Earth's surface, and varies with location of interest. We define it by unit axis vectors: x_n, y_n and z_n, where the x_n- and y_n-axis are located in the plane tangential to the Earth's surface at the location of interest, while z_n is perpendicular to this plane and thus points towards the Earth's center. Besides lying in the tangential plane, the x_n-axis is defined to always point towards the true north, which means that the y_n-axis ends up pointing roughly towards the east, therefore the name. In Figure 2.2 we show the NED frame at three different longitudes. The dashed line represents the z_n-axis.

2.1.4 Orbit Frame

The Orbit frame has its origin in the satellite's center of mass and its vector space is spanned by the unit axis vectors x_o, y_o and z_o. The axis z_o-is defined to always point towards the Earth's center (*nadir* direction) while the x_o-axis is defined to point along the satellite's linear velocity vector, tangential to its orbit. This places the y_o-axis normal to the orbital plane.

2.1.5 Body Frame

The Body frame too, has its origins in the center of the mass of the satellite, but it is in addition fixed to the satellite body and thus moves and rotates with the satellite. Body frame is defined by y_b-which is the axis of maximum[3] inertia and z_b-the axis of minimum inertia. As earlier we define the Body frame to be a right hand orthogonal coordinate system such that x_b-is found by the right hand rule.

2.2 Attitude Representation

The attitude of the satellite is most conveniently represented as a coordinate transformation (deviation) relative to a reference frame of choice. Now that the reference frames in the previous section have been defined, we need to develop the mathematical agility that enables us to seamlessly transform and rotate one frame into the other and back. This can be done in several ways. For our purposes, we present the concepts of the *rotation matrices*, *Euler parameters* and *quaternions.* As mentioned earlier, the theory presented here is in line with that in [4].

2.2.1 The Rotation Matrix

Let us start by introducing two arbitrary coordinate systems $\mathcal{F}_a$, $\mathcal{F}_b \in \mathbb{R}^3$, spanned by orthogonal unit vector sets $\vec{a_1}$, $\vec{a_2}$, $\vec{a_3}$ and $\vec{b_1}$, $\vec{b_2}$, $\vec{b_3}$ respectively. We also assume the existence of a vector $\mathbf{v}$ which can be represented with respect to any of the frames $\mathcal{F}_a$ and $\mathcal{F}_b$. We use notation:

[3]The reasoning for this can be found in 4.1, the 3-axes stabilization section.

$$\vec{v} = \sum_{i=1}^{3} v_i^a \vec{a_i} \text{ and } \vec{v} = \sum_{i=1}^{3} v_i^b \vec{b_i} \tag{2.1}$$

where $v_i^a = \vec{v} \cdot \vec{a_i}$ are the coordinates of $\vec{v}$ in $\mathcal{F}_a$, and $v_i^b = \vec{v} \cdot \vec{b_i}$ are the coordinates of $\vec{v}$ in $\mathcal{F}_b$. The column vectors in frame $\mathcal{F}_a$ and $\mathcal{F}_b$ become:

$$\mathbf{v}^a = \begin{bmatrix} v_1^a & v_2^a & v_3^a \end{bmatrix}^T \quad \text{and } \mathbf{v}^b = \begin{bmatrix} v_1^b & v_2^b & v_3^b \end{bmatrix}^T. \tag{2.2}$$

Combining (2.1) and (2.2) we are able to establish the relation between the coordinate vectors $\mathbf{v}^a$ and $\mathbf{v}^b$:

$$v_i^a = \vec{v} \cdot \vec{a_i} = (v_1^b \vec{b_1} + v_2^b \vec{b_2} + v_3^b \vec{b_3}) \cdot \vec{a_i} = \sum_{j=1}^{3} v_j^b (\vec{a_i} \cdot \vec{b_j}). \tag{2.3}$$

This leads to the following result:

$$\mathbf{v}^a = \mathbf{R}_b^a \mathbf{v}^b \quad \text{where } \mathbf{R}_b^a = \{\vec{a_i} \cdot \vec{b_j}\}. \tag{2.4}$$

We call $\mathbf{R}_b^a$ the rotation matrix from $\mathcal{F}_a$ to $\mathcal{F}_b$. We procede to present the properties of the rotation matrices. Proofs can be found in [4].

Property 1: $\mathbf{R}_{\mathbf{a}}^{\mathbf{b}} = (\mathbf{R}_b^a)^{-1} = (\mathbf{R}_b^a)^T$
Property 2: $\det \mathbf{R}_b^a = 1$.

Thus a special orthogonal group definition exists for the rotation matrices presented here: $SO(3) = \{\mathbf{R} | \mathbf{R} \in \mathbb{R}^{3x3},\ \mathbf{R}^T\mathbf{R} = \mathbf{I} \text{ and } \det(\mathbf{R}) = \mathbf{1}\}$.

Important results of the analysis done here are the two interpretations of the rotation matrix $\mathbf{R}_b^a$:

1. Let the vector $\vec{v}$ have the coordinate vector $\mathbf{v}^b$ in $\mathcal{F}_b$ and the coordinate vector $\mathbf{v}^a$ in $\mathcal{F}_a$. Then the rotation matrix $\mathbf{R}_b^a$ transforms the coordinate vector in $\mathcal{F}_b$ to the coordinate vector in $\mathcal{F}_a$ according to

$$\mathbf{v}^a = \mathbf{R}^a_b \mathbf{v}^b \ .$$

In this equation $\mathbf{R}^a_b$ acts as a coordinate transformation matrix. In layman's terms, this means that through multiplication $\mathbf{R}^a_b \mathbf{v}^b$ we can obtain the coordinates $\mathbf{v}^a$ in $\mathcal{F}_a$, of frameless vector $\vec{v}$ from its coordinates $\mathbf{v}^b$ in $\mathcal{F}_b$. From a third, arbitrary non-interacting coordinate system, the vectors $\mathbf{v}^a, \mathbf{v}^b, \vec{v}$ remain the same.$\square$

2. A vector $\vec{p}$ with coordinate vector $\mathbf{p}^a$ in $\mathcal{F}_a$ is rotated to the vector $\vec{q}$ with coordinate vector $\mathbf{q}^b = \mathbf{p}^a$ by

$$\mathbf{q}^a = \mathbf{R}^a_b \mathbf{p}^a.$$

In this equation $\mathbf{R}^a_b$ acts as a rotation matrix. In simpler terms, to help distinguish these two definitions: the rotation matrix rotates the coordinate vector $\mathbf{p}^a$ to a coordinate vector $\mathbf{q}^b$ that has the same coordinates relative to $\mathcal{F}_b$ as $\mathbf{p}^a$ has relative to $\mathcal{F}_a$ and then finally expresses it in $\mathcal{F}_a$.$\square$

Finally, an important property of the rotation matrices is that of composite rotations. A rotation from frame $\mathcal{F}_a$ to frame $\mathcal{F}_c$ may be described as a rotation from frame $\mathcal{F}_a$ to $\mathcal{F}_b$ followed by a rotation from $\mathcal{F}_b$ to $\mathcal{F}_c$. To illustrate: a vector $\vec{v}$ represented in $\mathcal{F}_a$ as $\mathbf{v}^a$ can be transformed to $\mathbf{v}^b$ in $\mathcal{F}_b$ by the following transformation$\mathbf{v}^b = \mathbf{R}^b_a \mathbf{v}^a$, and $\mathbf{v}^b$can then be transformed to $\mathbf{v}^c$ in $\mathcal{F}_c$ by$\mathbf{v}^c = \mathbf{R}^c_b \mathbf{v}^b$. This gives us the transformation matrix from $\mathcal{F}_c$ to $\mathcal{F}_a$:

$$\mathbf{v}^c = \mathbf{R}^c_b \mathbf{R}^b_a \mathbf{v}^a \ \rightarrow \ \mathbf{R}^c_a = \mathbf{R}^c_b \mathbf{R}^b_a.$$

2.2.2 Rotation Matrix Derivative

The time derivative of a rotation matrix gives us the angular velocity of a coordinate vector space. This is of importance for e.g. Body frame where it directly describes the satellites rotation velocity. We start off with the first property in the previous section:

$$\mathbf{R}^a_b = (\mathbf{R}^b_a)^{-1} = (\mathbf{R}^b_a)^T \implies \mathbf{R}^a_b (\mathbf{R}^a_b)^{-1} = \mathbf{R}^a_b (\mathbf{R}^a_b)^T = \mathbf{I}. \qquad (2.5)$$

Time differentiation of this product yields:

$$\frac{d}{dt}\left[\mathbf{R}_b^a(\mathbf{R}_b^a)^T\right] = \dot{\mathbf{R}}_b^a(\mathbf{R}_b^a)^T + \mathbf{R}_b^a(\dot{\mathbf{R}}_b^a)^T = \mathbf{0}. \tag{2.6}$$

By defining a matrix $\mathbf{S}$ as: $\mathbf{S} \triangleq \dot{\mathbf{R}}_b^a(\mathbf{R}_b^a)^T = \dot{\mathbf{R}}_b^a\mathbf{R}_a^b$, we can express (2.6) as $\mathbf{S} + \mathbf{S}^T = \mathbf{0}$, and it immediately follows that $\mathbf{S} = -\,\mathbf{S}^T$ which makes $\mathbf{S}$ skew-symmetric. Now, since any skew-symmetric 3×3 matrix can uniquely be expressed with 3 elements, we can define $\mathbf{S}(\boldsymbol{\omega}_{ab}^a)$ as an operator on a column vector $\boldsymbol{\omega}_{ab}^a$ that yields its skew-symmetric representation. Note that literature sometimes uses $(\boldsymbol{\omega}_{ab}^a)^\times$ to represent $\mathbf{S}(\boldsymbol{\omega}_{ab}^a)$.

$$\mathbf{S}(\boldsymbol{\omega}_{ab}^a) = (\boldsymbol{\omega}_{ab}^a)^\times = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}. \tag{2.7}$$

It turns out that the the vector $\boldsymbol{\omega}_{ab}^a$ has a fundamental physical interpretation: let the vector $\vec{\omega_{ab}}$ be defined by requiring that its coordinate form $\boldsymbol{\omega}_{ab}^a$ in frame $\mathcal{F}_a$ satisfies:

$$\mathbf{S}(\boldsymbol{\omega}_{ab}^a) = \dot{\mathbf{R}}_b^a(\mathbf{R}_b^a)^T. \tag{2.8}$$

The vector $\vec{\omega_{ab}}$ is then said to be the angular velocity vector of frame $\mathcal{F}_b$ relative to frame $\mathcal{F}_a$, which then makes the coordinate vector $\boldsymbol{\omega}_{ab}^a$ the angular velocity of frame $\mathcal{F}_b$ relative to frame $\mathcal{F}_a$ expressed in the coordinate frame $\mathcal{F}_a$.
We post-multiply (2.8) with $\mathbf{R}_b^a$, arriving at:

$$\mathbf{S}(\boldsymbol{\omega}_{ab}^a)\mathbf{R}_b^a = \dot{\mathbf{R}}_b^a(\mathbf{R}_b^a)^T\mathbf{R}_b^a = \dot{\mathbf{R}}_b^a \implies \dot{\mathbf{R}}_b^a = \mathbf{S}(\boldsymbol{\omega}_{ab}^a)\mathbf{R}_b^a. \tag{2.9}$$

By following the same line of though; using (2.7) we can rewrite (2.9) to arrive at the alternative form:

$$\dot{\mathbf{R}}_b^a = \mathbf{R}_b^a\mathbf{S}(\boldsymbol{\omega}_{ab}^b). \tag{2.10}$$

We can note that angular velocity has the property $\boldsymbol{\omega}_{ab}^a = -\boldsymbol{\omega}_{ba}^a$.

2.2.3 Euler Parameters and Quaternions

We start off briefly with Euler angles. With these we refer to both the roll-pitch-yaw angles and classical Euler angles. Both of these use a three parameter representation to define any rotation (orientation) of a frame by a composition of three *simple rotations*[4] around some of the principal axis of a frame. The downside to this very intuitive representation of any arbitrary rotation is that it suffers from possible singularities at e.g. $\cos(\frac{\pi}{2})$. This leads to numerical problems and in worst cases infeasibility when inverting a matrix with such ill conditioned elements. Euler parameters on the other hand use a four parameter representation of orientation. The advantage is to avoid the singularity problem that arise with Euler angles.

Let us start by stating that any rotation can be represented as a rotation of specific angle θ about a particular axis $\mathbf{k} = [k_1\, k_2\, k_3]^T$; for proof see section 6.6.2 in [4]. From this we define the Euler parameters in the following manner:

$$\eta = \cos(\frac{\theta}{2}) \text{ and } \boldsymbol{\epsilon} = [\epsilon_1\, \epsilon_2\, \epsilon_3] = \mathbf{k}\sin(\tfrac{\theta}{2}). \tag{2.11}$$

We note that

$$\eta^2 + \boldsymbol{\epsilon}^T\boldsymbol{\epsilon} = 1. \tag{2.12}$$

Through insertion of trigonometric identities and manipulation (See [4]) of (2.12), we are able to express the rotation matrix $\mathbf{R}_{\mathbf{k},\theta}$ in terms of Euler parameters η and $\boldsymbol{\epsilon}$:

$$\mathbf{R}_{\mathbf{k},\theta} = \mathbf{R}_e(\eta, \boldsymbol{\epsilon}) = \mathbf{I} + 2\eta\mathbf{S}(\boldsymbol{\epsilon}) + 2\mathbf{S}^2(\boldsymbol{\epsilon}). \tag{2.13}$$

It may be of interest that:

$$\mathbf{R}_e(\eta, \boldsymbol{\epsilon}) = \mathbf{R}_e(-\eta, -\boldsymbol{\epsilon}) \text{ and } \mathbf{R}_e(\eta, \boldsymbol{\epsilon})^T = \mathbf{R}_e(\eta, -\boldsymbol{\epsilon}). \tag{2.14}$$

[4]A rotation around the x-,y- or z-axis. E.g. $\mathbf{R}_x(\phi)$ - a rotation of ϕ radians around the principal x-axis.

To simplify things and enable us to apply a wealth of techniques and analysis tools, we exploit the fact that the vector we are dealing with, $\mathbf{p}=(\eta\,\boldsymbol{\epsilon}^T)^T$, is a *unit quaternion vector.* Thus quaternion theory may be applied. Basically a quaternion is a *hypercomplex number*[5], in our case a four dimensional one. A quaternion may be represented as a vector of the form $\mathbf{q} = \begin{bmatrix} \alpha & \boldsymbol{\beta}^T \end{bmatrix}^T$, where α represents the real scalar part and $\boldsymbol{\beta}^T$ represents a 1×3 dimensional imaginary vector part. We introduce the differential equations for the Euler parameters connecting them to the angular velocity of the satellite, $\boldsymbol{\omega}_{OB}^B$, in order to analyze attitude changes in time:

$$\dot{\eta} = -\frac{1}{2}\boldsymbol{\epsilon}^T \boldsymbol{\omega}_{OB}^B \tag{2.15}$$

$$\dot{\boldsymbol{\epsilon}} = \frac{1}{2}\left[\eta\mathbf{1} + \mathbf{S}(\boldsymbol{\eta})\right]\boldsymbol{\omega}_{OB}^B. \tag{2.16}$$

For proof and derivation see [4].

2.3 Transformations Between Frames

For later reference, we here derive the essential frame transformations. We make two assumptions: the eccentricity of the satellite's orbit is zero and the altitude of its orbit is 600km. However we do not, for theory development, assume the inclination to be exactly[6] 90°.

2.3.1 ECI to ECEF

The transformation from ECI to ECEF frame depends on the type of orbit the satellite assumes, e.g. a sun-synchronous polar orbit. It also depends on the specific parameters of the orbit and a specific, suitable definition of relative time. Since we at the present have very little information on the orbit, we choose to set the transformation matrix from ECI to ECEF to an identity matrix:

[5]We may say that complex numbers are a subgroup of hypercomplex numbers, just as real numbers are a subgroup of complex numbers. Although some special rules apply when manipulating quaternions

[6]We do not assume to have a perfect polar orbit.

$$\mathbf{R}_I^E = \mathbf{R}_E^I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.17}$$

We make this simplification to be able to manipulate frames seamlessly even with insufficient information. This simplification does not impact our prototype design. The exact transformation is however important for the final module, since satellite rotation about the earth is expressed relative to the ECI frame. For more information on a possible transformation matrix see Kristiansen, [12].

2.3.2 ECEF to Orbit Frame

This frame rotation can be represented as a composition of four simple rotations. Three of these are needed to account for every possible satellite orbit position, while the fourth rotation is just a static (constant) rotation[7]. The rotations appear in the following order:

1. Rotation θ about the z_e-axis.
2. Rotation $-\lambda$ about the new, current, y-axis.
3. Rotation $-90°$ about the new, current, y-axis.
4. Rotation δ about the new, current, z-axis.

Here θ is the longitude of the satellite, λ the latitude and δ the angle between the satellite's velocity vector and the $(\vec{z_m}, \vec{z_o})$ plane[8]. It is important to note in the above rotations that after each rotation the proceding rotation-axis changes. The complete rotation can be expressed as:

$$\mathbf{R}_O^E = \mathbf{R}_{z_e,\eta}\mathbf{R}_{y,-\lambda}\mathbf{R}_{y,-90}\mathbf{R}_{z,\delta} \tag{2.18}$$

[7]These can easily be compressed into three rotations but we choose to keep them separated for clarity.

[8]We define $\vec{z_m}$ as the vector pointing towards the magnetic north pole, but for the dipole geomagnetic field we set $\vec{z_m}$ and $\vec{z_e}$ identical each other, thus simplifying the δ variable.

In Figure (2.3), we provide an illustration to shed some light on this somewhat tedious rotation. Note that the last (δ) rotation is not included as we did not manage to create figure depicting it clearly. The θ rotation is done in the negative direction to obtain a simpler figure:

$$\begin{aligned}\mathbf{R}_O^E &= \begin{bmatrix} c\theta & -s\theta & 0 \\ s\theta & c\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c\lambda & 0 & -s\lambda \\ 0 & 1 & 0 \\ s\lambda & 0 & c\lambda \end{bmatrix} \begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} c\delta & -s\delta & 0 \\ s\delta & c\delta & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} -c\theta s\lambda c\delta - s\theta s\delta & c\theta s\lambda s\delta - s\theta c\delta & -c\theta c\lambda \\ c\theta s\delta - s\theta s\lambda c\delta & c\theta c\delta + s\theta s\lambda s\delta & -s\theta c\lambda \\ c\lambda c\delta & -c\lambda s\delta & -s\lambda \end{bmatrix}. \end{aligned} \tag{2.19}$$

We use notation $s\circ$ to denote $\sin(\circ)$ and $c\circ$ to denote $\cos(\circ)$. In the second matrix that we exploit the fact that $\cos(-\lambda) = \cos(\lambda)$ and $\sin(-\lambda) = -\sin(\lambda)$. From here we arrive at our destination through a rotation matrix property:

$$\mathbf{R}_E^O = (\mathbf{R}_O^E)^T = \begin{bmatrix} -c\theta s\lambda c\delta - s\theta s\delta & c\theta s\delta - s\theta s\lambda c\delta & c\lambda c\delta \\ c\theta s\lambda s\delta - s\theta c\delta & c\theta c\delta + s\theta s\lambda s\delta & -c\lambda s\delta \\ -c\theta c\lambda & -s\theta c\lambda & -s\lambda \end{bmatrix}. \tag{2.20}$$

2.3.3 NED to ECEF

The rotation to and from NED frame is be used directly in the control module. The attitude provided by the determination module, under development by Kristian Jenssen and Kaan Yabar, is however represented in the NED frame. Thus, in order to combine the two systems we need to have a rotation matrix transforming the attitude from NED frame to the Orbit/Body frames - which happens via the ECEF frame.

The rotation from NED to ECEF can be represented as a composition of two rotations, the first being a rotation θ about the z_n axis and a second rotation $-\left(\frac{\pi}{2} + \lambda\right)$ about the new y_n axis, where θ is the longitude and λ the latitude of the satellite. Mathematically this is expressed as:

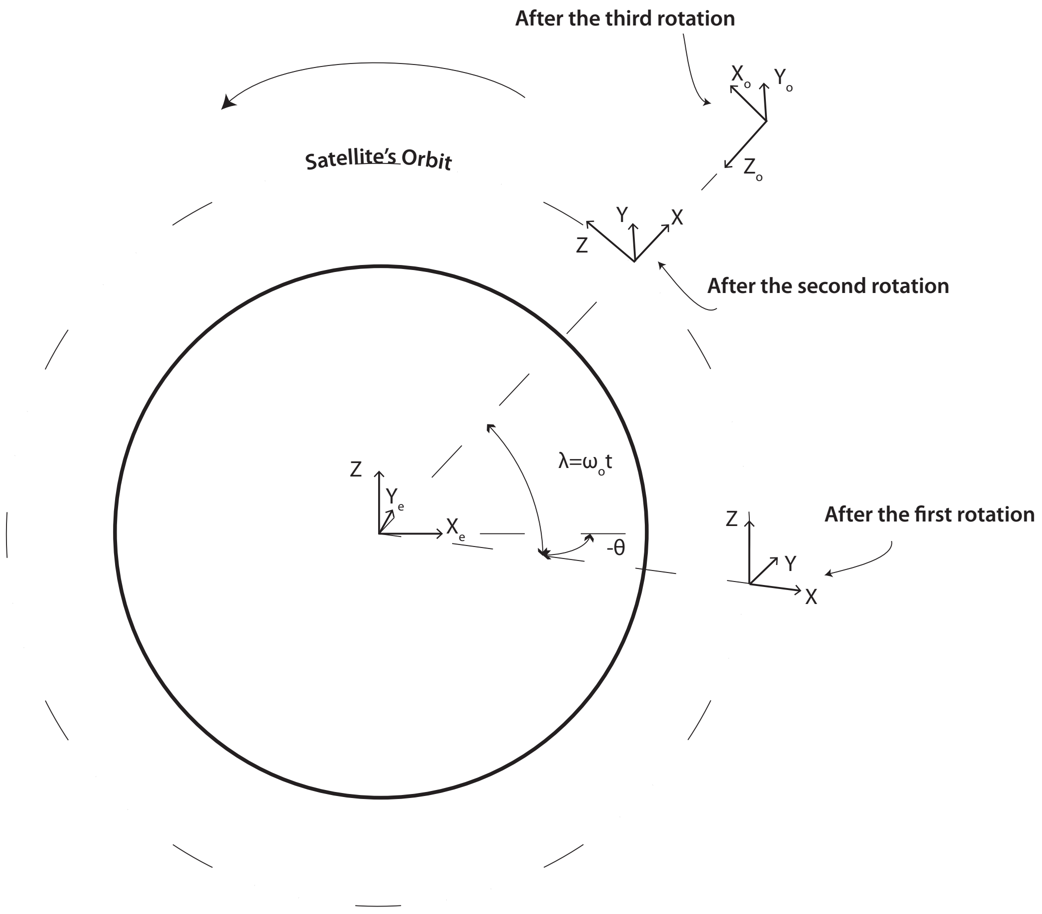


Figure 2.3: Illustration of the ECEF to Orbit frame transformation.

$$\mathbf{R}_E^N = \begin{bmatrix} -s\lambda & 0 & c\lambda \\ 0 & 1 & 0 \\ -c\lambda & 0 & -s\lambda \end{bmatrix} \begin{bmatrix} c\theta & s\theta & 0 \\ -s\theta & c\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -s\lambda c\theta & -s\lambda s\theta & c\lambda \\ -s\theta & c\theta & 0 \\ -c\lambda c\theta & -c\lambda s\theta & -s\lambda \end{bmatrix} \tag{2.21}$$

where we use the trigonometric identities for a $\frac{\pi}{2}$ shift, such as: $\sin(\lambda + \frac{\pi}{2}) = \cos\lambda$ and $\cos(\lambda + \frac{\pi}{2}) = -\sin\lambda$. This gives us the NED to ECEF rotation matrix:

$$\mathbf{R}_N^E = \left(\mathbf{R}_E^N\right)^T = \begin{bmatrix} -s\lambda c\theta & -s\theta & -c\lambda c\theta \\ -s\lambda s\theta & c\theta & -c\lambda s\theta \\ c\lambda & 0 & -s\lambda \end{bmatrix}. \tag{2.22}$$

2.3.4 Orbit to Body Frame

Once the quaternions (Euler parameters) for the body frame are available the frame transformation is obtained directly out of the theory presented in (2.2.3). From (2.13) we see that:

$$\mathbf{R}_B^O = \mathbf{I} + 2\eta\mathbf{S}(\boldsymbol{\epsilon}) + 2\mathbf{S}^2(\boldsymbol{\epsilon}) \tag{2.23}$$

and finally:

$$\mathbf{R}_O^B = (\mathbf{R}_B^O)^T = \begin{bmatrix} \mathbf{c}_1 & \mathbf{c}_2 & \mathbf{c}_3 \end{bmatrix}. \tag{2.24}$$

.

We include the last columnwise notation, as we later make use of the column vectors in $\mathbf{R}_O^B$.

2.4 Stability

We present the necessary Lyapunov stability theorems which will be applied in the later sections. These are extracted from Chapter 4.1 in Nonlinear

Systems by Hassan K. Khalil [11].

Theorem 1: Let $x = 0$ be an equilibrium point for $\dot{x} = f(x)$ and $D \subset R^n$ be a domain containing $x = 0$. Let $V : D \rightarrow R$ be a continuously differentiable function such that

$$V(0) = 0 \text{ and } V(x) > 0 \text{ in } D - \{0\} \tag{2.25}$$

$$\dot{V}(x) \leq 0 \text{ in } D. \tag{2.26}$$

Then, $x = 0$ is stable. Moreover, if

$$\dot{V}(x) < 0 \text{ in } D - \{0\} \tag{2.27}$$

then $x = 0$ is asymptotically stable $\Diamond$.

Theorem 2: Let $x = 0$ be an equilibrium point for $\dot{x} = f(x)$. Let $V : R^n \rightarrow R$ be a continuously differentiable function such that

$$V(0) = 0 \text{ and } V(x) > 0,\ \forall x \neq 0 \tag{2.28}$$

$$||x|| \rightarrow \infty \Rightarrow V(x) \rightarrow \infty \tag{2.29}$$

$$\dot{V}(x) < 0,\ \forall x \neq 0 \tag{2.30}$$

then $x = 0$ is globally asymptotically stable $\Diamond$.

Remark 1: For a system with an equilibrium in $x \neq 0$ we may perform a change of variables as: $y = x - x_e$, where x_e is the equilibrium.
Remark 2: Theorem 2 differs from the first at two points: the domain D in Theorem 2 is the whole state space and $V(x)$ is radially unbounded.
Remark 3: We present only the time invariant Lyapunov theorems. This is because the geomagnetic field as the only time-varying component of the system is varying at a relatively negligable rate. For time varying systems we would refer to uniform stability when choice of t_0 does not effect system stability.

Chapter 3

Mathematical Modelling

The mathematical models for the the relevant dynamics appear to mostly have been developed and collected by Per Kolbjørn Soglo in [20] from books such as [7], then further refined in [5, 12] and neatly summed up by Jan Tommy Gravdahl in [6]. Thus the modelling presented here follows these texts closely.

3.1 Gravitational Torque

Gravitational torques are fundamental to the attitude dynamics of spacecrafts. An object with a non-uniform, non-symmetrically positioned mass distribution (w.r.t. the gravity vector), and exposed to a quadratically diminishing gravity field will be affected by a torque - gravitational torque. In other words, because the center of gravity and the center of mass in practice never exactly overlap, everything is exposed to this gravitational torque. According to Chapter 8.1 in [7], when developing a simple model for the gravitational torque it is convenient to make four simple assumptions:

1. Only one celestial primary is considered - i.e. we ignore the gravitational fields from all other objects besides the Earth, such as the Moon, Sun and self-gravity.

2. The considered primary (Earth) possesses a spherically symmetrical mass distribution.

3. The spacecraft is small compared to its distance to the mass center of the primary.

4. The spacecraft consists of a single body.

As we see, these assumptions are not all that conservative, as expanding the model to account for all the imperfections would only slightly increase it's accuracy. With these assumptions in place we are ready to develop the equations we need. Using the Newton's law of gravity we start off with the force that an element dm in the satellite is exposed to:

$$\vec{df} = -\mu \frac{\vec{R}}{R^3} dm \tag{3.1}$$

where $\mu = G_e m_e = 3.986 \cdot 10^{14}$, G_e is the Earth's gravitational constant, m_e the Earth's mass, $\vec{R} = \vec{R_c} + \vec{r}$ the vector to the element dm as shown in Figure 3.1, and $R = |\vec{R}|$ the absolute distance of dm from Earth's center. Further we have that the torque from dm about satellite's center of mass is:

$$\vec{dg_c} = \vec{r} \times \vec{df}. \tag{3.2}$$

From this we can obtain the total gravitational torque on the satellite by integrating over its entire body:

$$\vec{g_c} = -\mu \int_B \frac{\vec{r} \times \vec{R}}{R^3} dm. \tag{3.3}$$

In addition we can express the satellite's potential energy as:

$$U = -\mu \int_B \frac{dm}{R}. \tag{3.4}$$

Due to our third assumption $\frac{r}{R_c} \ll 1$ we are able to apply a binomial expansion:

$$R^{-3} = R_c^{-3} \left[1 - \frac{3(\vec{r} \cdot \vec{R_c})}{R_c^2} + O(\frac{r^2}{R_c^2}) \right] \tag{3.5}$$

where $O(\circ)$ denotes the higher order terms. This gives us:

$$R^{-1} = R_c^{-1} \left[1 - \frac{\vec{r} \cdot \vec{R}_c}{R_c^2} - \frac{1}{2}\frac{r^2}{R_c^2} + \frac{3}{2}\frac{(\vec{r} \cdot \vec{R}_c)^2}{R_c^4} + O(\frac{r^3}{R_c^3}) \right]. \tag{3.6}$$

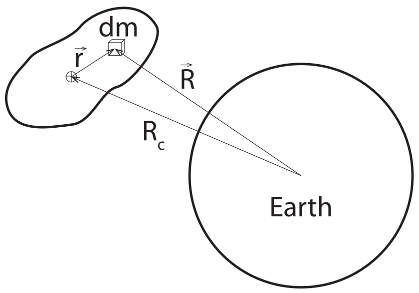


Figure 3.1: Gravitational forces working on the satellite. Inspired by [7].

By inserting for $\vec{R}$ and 3.5 into 3.3 we obtain:

$$\vec{g_c} = -3\left(\frac{3\mu}{R_c^5}\right)\vec{R_c} \times \int_B \vec{r}\vec{r}dm \cdot \vec{R_c}. \tag{3.7}$$

Exploiting the fact that the second moment of inertial about a center of mass is:

$$\vec{I} \triangleq \int_B (r^2\vec{1} - \vec{r}\vec{r})dm \tag{3.8}$$

where $\vec{1}$ represents the unit dyadic[1]. We can rewrite the gravitational torque as:

$$\vec{g_c} = 3\left(\frac{\mu}{R_c^3}\right)\vec{z}_o \times \vec{I} \cdot \vec{z}_o \tag{3.9}$$

where $\vec{z}_o$ is the z-axis in the orbit frame which can mathematically be defined as $\vec{z}_o = \frac{\vec{R}_c}{R_c}$. Similarly by inserting 3.6 into 3.4 we arrive at the potential energy of the satellite:

$$U = -\frac{\mu m}{R_c} - \frac{1}{2}\frac{\mu}{R_c^3} \cdot trace(\bar{I}) + \frac{3}{2}\frac{\mu}{R_c^3}\vec{z}_o \cdot \bar{I} \cdot \vec{z}_o \tag{3.10}$$

where we exploit the fact that $\int r^2 dm = \frac{1}{2}trace(\bar{I})$. We can finally arrive at the gravitational torque and potential energy expressed in the body frame, using the vectricies[2] definition $\mathbf{I} \triangleq \mathcal{F}_b \cdot \bar{I} \cdot \mathcal{F}_b^T$ and the equality $\vec{z}_o = \mathcal{F}_b^T\mathbf{c_3}$:

[1]Interested reader may look to [4] for explanation of dyadics.

[2]Explained and introduced in [7]

$$\mathbf{g}_c^B = 3(\frac{\mu}{R_c^3})\mathbf{c_3} \times \mathbf{I}\mathbf{c_3} \tag{3.11}$$

$$U = -\frac{\mu m}{R_c} - \frac{1}{2}\frac{\mu}{R_c^3} \cdot trace(\mathbf{I}) + \frac{3}{2}\frac{\mu}{R_c^3}\mathbf{c_3^T}\mathbf{I}\mathbf{c_3} \tag{3.12}$$

where $\mathbf{c_3}$ is defined in 2.24.
We assume to position the $\mathcal{F}_b$ frame in the satellite such that the inertia matrix $\mathbf{I}$ becomes:

$$\mathbf{I} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}. \tag{3.13}$$

3.2 Dynamical Satellite Attitude Model

A satellite's dynamics can be represented as those of a rigid body. From Ch.3 in [7] together with the papers mentioned, we shortly arrive at a dynamic equation for the satellite's momentum:

$$\mathbf{I}\dot{\boldsymbol{\omega}}_{\mathbf{IB}}^{\mathbf{B}} \quad = \quad \boldsymbol{\tau}^B - \boldsymbol{\omega}_{IB}^B \times (\mathbf{I}\boldsymbol{\omega}_{IB}^B) \tag{3.14}$$

where $\mathbf{I}$ is the satellite's moment of inertia, $\boldsymbol{\omega}_{IB}^B$ is the angular velocity of the body frame relative to inertial frame expressed in the body frame and $\boldsymbol{\tau}^B$ is the sum of all torques working on the satellite. The angular velocity $\boldsymbol{\omega}_{IB}^B$ can be expressed as a composition of two rotations:

$$\boldsymbol{\omega}_{IB}^B = \boldsymbol{\omega}_{OB}^B + \boldsymbol{\omega}_{IO}^B = \boldsymbol{\omega}_{OB}^B + \mathbf{R}_O^B\boldsymbol{\omega}_{IO}^O = \omega_{OB}^B - \mathbf{c_2}\omega_o \tag{3.15}$$

where angular velocity $\boldsymbol{\omega}_{OI}^O$ has a simple form (this is a rotation about the y_o axis; try to visualize it):

$$\boldsymbol{\omega}_{OI}^O = \begin{bmatrix} 0 & -\omega_o & 0 \end{bmatrix}^T \tag{3.16}$$

where $\omega_o = \sqrt{\frac{\mu}{R_c^3}}$. The scalar ω_o is obtained by equalling Newton's 2. law for uniform circular motion to the law of universal gravitation.

$$a = \frac{v_o^2}{R_c} \quad \rightarrow \quad m_B \frac{v_o^2}{R_c} = F = G_e \frac{m_e m_B}{R_c^2} \tag{3.17}$$

$$v_o = \sqrt{\frac{\mu}{R_c}} \quad \rightarrow \quad \omega_o = \frac{\sqrt{\frac{\mu}{R_c}}}{R_c} = \sqrt{\frac{\mu}{R_c^3}}. \tag{3.18}$$

The total torque $\boldsymbol{\tau}^B$ in 3.14 can be split up as:

$$\boldsymbol{\tau}^B = \boldsymbol{\tau}_g^B + \boldsymbol{\tau}_m^B \tag{3.19}$$

where $\boldsymbol{\tau}_g^B$ is the gravitational torque on the satellite and $\boldsymbol{\tau}_m^B$ is the torque produced by the magnetic coils. We set $\boldsymbol{\tau}_g^B = \mathbf{g}_c^B$ from Section 3.1 and develop $\boldsymbol{\tau}_m^B$ in the next section.

3.3 Magnetic Torque

Here we present two models for the Earth's magnetic field: a simplistic dipole geomagnetic model and a more sophisticated and accurate International Geomagnetic Reference Field (IGRF-11) model. We also derive and present the equations for the magnetic coil actuators installed on the satellite in order to control its attitude.

3.3.1 The Geomagnetic Field Models

The geomagnetic field model to be implemented on the satellite is obviously of central importance for the attitude control. The better the model, taking into account the computational time and required power, the better the control ultimately will be. Nevertheless we start of with a simple dipole model as proposed in [20], and then move on to the IGRF-11 model and its equations presented in [12].

3.3.1.1 Dipole Model

Dipole model is a simple symmetrical model of the geomagnetic field, that completely ignores the longitudinal components of the magnetic field vector and can thus simply be illustrated two-dimensionally, as in Figure 3.2. We note that the geographic and magnetic poles do not (necessarily) overlap, in fact the magnetic poles wander constantly relative to the geographic poles

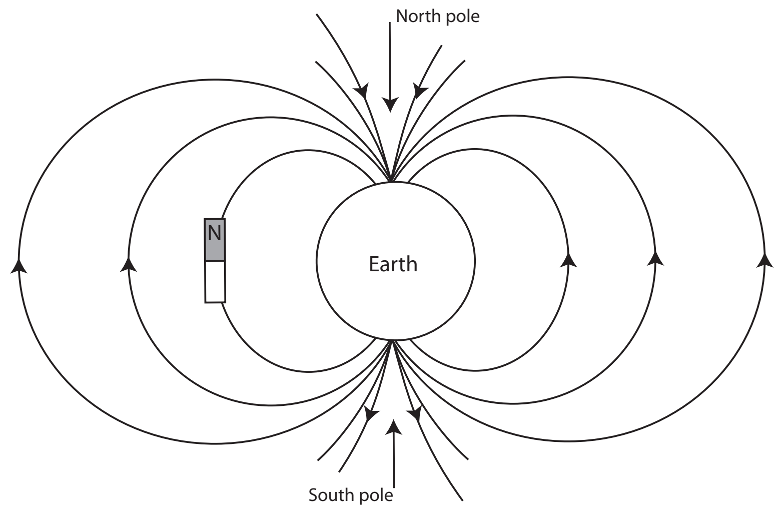


Figure 3.2: Dipole model.

- but for control simulation purposes with an already imperfect model, we ignore this fact.

The dipole model given in the orbit frame, is extracted from [20]:

$$\mathbf{B}^O = B_0 \begin{bmatrix} \cos\lambda\cos\delta \\ -\cos\lambda\sin\delta \\ 2\sin\lambda \end{bmatrix} \tag{3.20}$$

where $B_0 = \frac{\mu}{R_c^3}$, λ is the latitude, and δ the angle between the satellite's velocity vector and the $(\vec{z_i}, \vec{z_o})$ plane. The time derivative of this field is:

$$\dot{\mathbf{B}}^O = \omega_o B_0 \begin{bmatrix} -\sin\lambda \\ 0 \\ 2\cos\lambda \end{bmatrix}. \tag{3.21}$$

3.3.1.2 International Geomagnetic Reference Field

The IGRF model estimated every 5 years by the International Association of Geomagnetism and Aeronomy (IAGA), is essentially a set of Gaussian coefficients g_n^m and h_n^m, that can be used in a spherical harmonic[3] model to

[3]Spherical harmonics may be seen as an extension of a 2 dimensional Taylor approximation to approximate 3 dimensional phenomena.

approximate the Earth's magnetic field. Because the geomagnetic field is time-varying, the IGRF model also includes the so called secular variation (SV) coefficients such that for the 5 years after the most recent epoch, the SV can be used for forward linear extrapolation. The IGRF-11 model is the 11th generation model which gives the 2010 spherical harmonic coefficients together with their respective SV. These coefficients are extracted from [16] and can be found in Table B.1. For obvious reasons, this model describes the geomagnetic field much more accurately then the naive dipole model. The model is somewhat complex and both its theoretical and implementational aspects have already been studied thoroughly in [12, 14] and especially well explained in [3]. We therefore only briefly present the main IGRF equation for reference, extracted from [3], which is the negative gradient of a scalar potential function modelled by spherical harmonics:

$$\begin{aligned}\mathbf{B} &= -\nabla V(R_c, \lambda^{'}, \theta) \\ &= \nabla \left\{ R_e \sum_{n=1}^{k} \left(\frac{R_e}{R_c}\right)^{n+1} \sum_{m=0}^{n} [g_n^m cos(m\theta) + h_n^m sin(m\theta)]] P_n^m(\lambda^{'}) \right\}\end{aligned} \tag{3.22}$$

where $R_e = 6371.2{\cdot}10^3$m is the Earth radius, R_c the distance from the Earth's center to the satellite, and $P_n^m(\lambda^{'})$ the Schmidt normalized associated Legendre polynomials. From 3.22 we can see that the magnetic field, obviously, depends on distance R_c, co-latitude $\lambda^{'}$ and longitude θ. Thus taking the gradient of $V(R_c, \lambda^{'}, \theta)$ gives us the geomagnetic field vector components in spherical coordinates, represented in the ECEF frame. The final equations for the three vector components are:

$$\begin{aligned}B_r &= -\frac{\partial V}{\partial R_c} \\ &= \sum_{n=1}^{k} \left(\frac{R_e}{R_c}\right)^{n+2} (n+1) \sum_{m=0}^{n} [g^{m,n} cos(m\theta) + h^{n,m} sin(m\theta)]\, P_n^m(\lambda^{'})\end{aligned} \tag{3.23}$$

$$\begin{aligned}B_{\lambda'} &= -\frac{1}{R_c}\frac{\partial V}{\partial \lambda'} \\ &= -\sum_{n=1}^{k} \left(\frac{R_e}{R_c}\right)^{n+2} \sum_{m=0}^{n} [g^{m,n} cos(m\theta) + h^{n,m} sin(m\theta)]\, \frac{\partial P^{n,m}(\lambda^{'})}{\partial \lambda'}\end{aligned} \tag{3.24}$$

$$\begin{aligned} B_\theta &= -\frac{1}{R_c \cdot \sin(\lambda')}\frac{\partial V}{\partial \theta} \\ &= -\frac{1}{\sin(\lambda')}\sum_{n=1}^{k}\left(\frac{R_e}{R_c}\right)^{n+2}\sum_{m=0}^{n} m\left[-g^{m,n} sin(m\theta) + h^{n,m} sin(m\theta)\right] P_n^m(\lambda') \end{aligned} \tag{3.25}$$

For illustrative insight, Figure 2.9 in [14] can provide the reader with basic intuition on the nature of spherical harmonics.

3.3.2 Magnetic Coil Model

It is well known that a current flowing through a conductor in a loop generates a dipole magnetic field. This magnetic dipole is a magnetic moment which can be described as a vector. The vector is perpendicular to the loop and points in the same direction as the extended thumb of one's right hand when the fingers are wrapped around the loop in the direction of the current - therefore the right-hand rule. The magnitude of the moment generated by a single loop is relatively tiny, but is also proportional to the number of turns. By stacking turns on top of each other we construct a winding to arrive at the concept of a magnetic coil. For multiple turns we can still apply the right hand rule in order to find the moment's direction[4]. The magnitude on the other hand, for the coils mounted on the x^+-, y^+- and z^+-faces of the satellite can be expressed as:

$$\begin{bmatrix} m_x \\ m_y \\ m_z \end{bmatrix} = \begin{bmatrix} N_x \cdot A_x \cdot i_x \\ N_y \cdot A_y \cdot i_y \\ N_z \cdot A_z \cdot i_z \end{bmatrix} \tag{3.26}$$

where N is the number of turns in the coil, A the area enclosed by the coil and i the current flowing through the coil. The cross product of this magnetic moment and the moment of the geomagnetic field generate a torque with which the satellite can be maneuvered. The three magnetic coils installed on the satellite are placed perpendicular to one another in order to gain the highest degree of controllability.

[4]E.g. for a solenoid the north magnetic pole will be at the end where the current exits the coil.

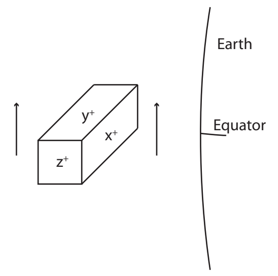


Figure 3.3: Loss of controllability.

By its nature, a coil can only produce a moment in one direction; more specifically a moment perpendicular to the area it encloses. This obviously sets restrictions on how the control algorithms have to be created, as the direction of the desired, optimal moment, will rarely coincide with the any single coil moment vector. Thus the desired moments have to be decomposed and split between the three coils. The biggest issue however, comes from the way the torque is generated. It is trivial that the cross product of two vectors is the largest when the two vectors are perpendicular to each other and zero if the vectors are parallel. Assuming a dipole geomagnetic model we will attempt to illustrate this problem through Figure 3.3. In the figure the satellite is above the equator with its x^+ face orientated towards earth. Lets now say that the operator wishes to rotate the satellite such that its z^+ face points towards the earth. To obtain the desired orientation we need to apply a torque about the y axis. Sending a current through the x- or z-face coils creates torques about the z- and x-axis respectively, while sending a current through the y-face coil generates no torque at all because the y-axis is parallel to the earth's magnetic field. Thus if the satellite was to reach such a state, all control would be lost. Luckily this scenario is only theoretical due gravitational torque, varying geomagnetic field and different kinds of disturbances the satellite is exposed to. In practice we only need to be aware of the problem in order to save battery life when the body and geomagnetic frames are ill-aligned.

Chapter 4

Control

In this chapter we analyze the satellite-earth system to find the satellite's requirements for achieving both 3-axes and 1-axis stabilization. We then move on to develop the nonlinear controllers necessary for this stabilization. We first develop a detumbling controller and finally a reference controller for large deviations, both to be used on the prototype and eventually on the final module. Finally we introduce and explain some basics concerning the practicalities when executing such control, such as Pulse Width Modulation (PWM). For the stabilization requirements and control algorithms we look at the work done by Soglo, who elegantly derived these equations in his thesis [20].

4.1 Requirements for 3-axes Stabilization

Before discussing any control strategies we need to analyze the system equations with zero input, e.g. $\boldsymbol{\tau}_m = \mathbf{0}$. From Section 3.1, we have that whenever $I_x \neq I_y \neq I_z$ the satellite will be exposed to gravity torque, however, there exist certain satellite attitudes at which the satellite is in equilibrium. At these attitudes the gravity gradient will exert zero torque on the satellite. Peter C. Hughes argues in Chapter 9.2 in [7] that there exist 24 such equilibria. Each of the principal axes can point either towards the Earth's center or away from it; and, for each of these, the remaining two axes can either point along the orbit normal or opposite of the orbit normal. To investigate the required properties of the satellite's inertia that make one or more of the equilibria stable we apply Lyapunov theory to analyze the system and attempt to draw useful conclusions.

We start with the satellite's potential energy expressed in 3.12. By inserting for ω_o we arrive at:

$$U = -\frac{\mu m_B}{R_c} - \frac{1}{2}\omega_o^2(I_x + I_y + I_z) + \frac{3}{2}\omega_o^2\mathbf{c}_3^T\mathbf{I}\mathbf{c_3}. \tag{4.1}$$

Satellite's kinetic energy on the other hand, assumes a relatively simple form, as a sum of its translational and rotational kinetic energy:

$$T = T_{trans} + T_{rot} = \frac{1}{2}m_B\omega_o^2R_c^2 + \frac{1}{2}(\boldsymbol{\omega}_{IB}^B)^2\mathbf{I}\boldsymbol{\omega}_{IB}^B. \tag{4.2}$$

By inserting for $\boldsymbol{\omega}_I^B$ from 3.15 we arrive at:

$$T = T_0+T_1+T_2 = \frac{1}{2}m_B\omega_o^2R_c^2+\frac{1}{2}\omega_o^2\mathbf{c}_2^T\mathbf{I}\mathbf{c}_2-\omega_o\mathbf{c}_2^T\mathbf{I}\boldsymbol{\omega}_{OB}^B+\frac{1}{2}(\boldsymbol{\omega}_{OB}^B)^T\mathbf{I}\boldsymbol{\omega}_{OB}^B \tag{4.3}$$

where

$$T_0 = \frac{1}{2}m_B\omega_o^2R_c^2 + \frac{1}{2}\omega_o^2\mathbf{c}_2^T\mathbf{I}\mathbf{c}_2 \tag{4.4}$$
$$T_1 = -\omega_o\mathbf{c}_2^T\mathbf{I}\boldsymbol{\omega}_{OB}^B \tag{4.5}$$
$$T_2 = \frac{1}{2}(\boldsymbol{\omega}_{OB}^B)^T\mathbf{I}\boldsymbol{\omega}_{OB}^B. \tag{4.6}$$

As shown in [7] we can from here define an energy function H, which will help us arrive at a suitable Lyapunov function:

$$\begin{aligned} H &= T_2 - T_0 + U \\ &= \frac{1}{2}(\boldsymbol{\omega}_{OB}^B)^T\mathbf{I}\boldsymbol{\omega}_{OB}^B + \frac{3}{2}\omega_o^2\mathbf{c}_3^T\mathbf{I}\mathbf{c_3} - \frac{1}{2}\omega_o^2\mathbf{c}_2^T\mathbf{I}\mathbf{c}_2 \\ &\quad -\frac{1}{2}m_B\omega_o^2R_c^2 - \frac{\mu m_B}{R_c} - \frac{1}{2}\omega_o^2(I_x + I_y + I_z). \end{aligned} \tag{4.7}$$

Because a Lyapunov equation has to be zero when the state vector equals zero and we wish to analyze the equilibrium state where $\boldsymbol{\omega}_{OB}^{B} = 0$ and $R_{B}^{O} = \mathbf{1}$ we need to substract the energy, $H_0 \triangleq H|_{w_{OB}^{B}=0,\, R_{B}^{O}=\mathbf{1}}$, from H to reach the final Lyapunov equation. We end up with:

$$V \triangleq H - H_0 = \frac{1}{2}(\boldsymbol{\omega}_{OB}^{B})^T \mathbf{I}\boldsymbol{\omega}_{OB}^{B} + \frac{3}{2}\omega_o^2 \mathbf{c}_3^T \mathbf{I}\mathbf{c_3} - \frac{1}{2}\omega_o^2 \mathbf{c}_2^T \mathbf{I}\mathbf{c}_2 + \frac{1}{2}\omega_o^2 (I_y - 3I_z). \quad (4.8)$$

Soglo points out in [20], that it is usefull to write 4.8 in its scalar form, and use the rotation matrix properties $c_{12}^2 + c_{22}^2 + c_{32}^2 = 1$ and $c_{13}^2 + c_{23}^2 + c_{33}^2 = 1$, to reach:

$$\begin{aligned} V &= \frac{1}{2}(\boldsymbol{\omega}_{OB}^{B})^T \mathbf{I}\boldsymbol{\omega}_{OB}^{B} + \frac{3}{2}\omega_o^2 \left[(I_x - I_z)c_{13}^2 + (I_y - I_z)c_{23}^2\right] \\ &+ \frac{1}{2}\omega_o^2 \left[(I_y - I_x)c_{12}^2 + (I_y - I_z)c_{32}^2\right]. \end{aligned} \quad (4.9)$$

Now we can define a state vector $\mathbf{x} = \left[\begin{array}{ccccc} (\boldsymbol{\omega}_{OB}^{B})^T & c_{13} & c_{23} & c_{12} & c_{32} \end{array}\right]^T$, and finally examine the system's stability properties based on different moments of inertia. To assert any kind of stability of $\mathbf{x} = 0$ through Lyapunov stability theorems, we need V to be positive definite. It is easy to see from 4.9 that this is true for $I_y > I_x > I_z$. Our Lyapunov function has to satisfy the decrescent property as well - we have to examine its time derivative.
Taking the time derivative of 4.8 and manipulating it[1] we arrive at:

$$\dot{V} = (\boldsymbol{\omega}_{OB}^{B})^T \boldsymbol{\tau}_m^B. \quad (4.10)$$

But as we mentioned at the start of our derivation, we are examining the system with zero input, e.g. $\boldsymbol{\tau}_m = \mathbf{0}$. From this we can see that for $\mathbf{x} = \left[\begin{array}{ccccc} (\boldsymbol{\omega}_{OB}^{B})^T & c_{13} & c_{23} & c_{12} & c_{32} \end{array}\right]^T = \mathbf{0}$, with $\dot{V} \equiv 0$, the system satisfies the decrescent condition $\dot{V}(x) \leq 0$, and according to Chapter 9.2 in [7], is stable in Lyapunov sense. Now, using orthonormality property of rotation matrices: $c_{i1}^2 + c_{i2}^2 + c_{i3}^2 = 1$ and $c_{1i}^2 + c_{2i}^2 + c_{3i}^2 = 1$ we can examine a generic rotation matrix, to see whether $c_{13} = c_{23} = c_{12} = c_{32} = 0$ defines a unique state.

$$R_O^B = \begin{bmatrix} c_{11} & 0 & 0 \\ c_{21} & c_{22} & 0 \\ c_{23} & 0 & c_{33} \end{bmatrix} \quad (4.11)$$

[1]See Chapter 7 in [20] for proper derivation.

From the orthonormal property it immediately follows that $c_{21} = c_{23} = 0$. Exploiting yet another rotation matrix property, $\det(R) = 1$, we get that $c_{11}c_{22}c_{33} = 1$, leaving us with four stable attitudes; relative to the orbit frame:

$$R_O^B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R_O^B = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$$R_O^B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \quad R_O^B = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix}. \tag{4.12}$$

These are the four attitudes where $x_b || x_o$ and $z_b || z_o$. If we however have $I_x = I_y$, given the satellite's symmetry, we end up with the state vector $\mathbf{x} = \begin{bmatrix} (\boldsymbol{\omega}_{OB}^B)^T & c_{13} & c_{23} & c_{32} \end{bmatrix}^T$ and the generic matrix:

$$R_O^B = \begin{bmatrix} c_{11} & c_{12} & 0 \\ c_{21} & c_{22} & 0 \\ c_{23} & 0 & c_{33} \end{bmatrix} \tag{4.13}$$

from which we can only deduce that $c_{23} = 0$, giving us an infinite set of stable attitudes - all attitudes where $z_b || z_o$. Simpy put, this is 1-axis stabilization.

4.2 Detumbling Controller

A small experimental satellite of this type is carried to its orbit only as a companion of a much larger, commercial satellite. In addition each of these commercial launches carries multiple CubeSats. Once a certain altitude is reached, the small satellites are ejected in succession one after the other by a spring separation mechanism on the carrier rocket. Depending on the rocket's orientation and obvious design imperfections, the initial orientation and angular velocity of the satellite are as good as arbitrary. In other words, there is no way of knowing these in advance. This initial angular velocity is often in the literature referred to as *tumbling*. Removing (dissipating) this rotational energy from the system is thus called *detumbling*.

Below we present two detumbling controllers: a dissipative and a B-dot detumbling controller. Our focus is mainly on the dissipative controller because the data for its computation will be readily available from the attitude determination module, while the second, B-dot detumbling controller, requires rather heavy computation of the IGRF field's time derivative.

4.2.1 Dissipative Controller

Our primary control objective is to orient the satellite's camera mounted on one of the two 10×10cm faces to point towards the Earth. In order to do this, the control system needs first to dissipate the satellite's angular velocity relative to the Orbit frame. Note that having zero angular velocity of Body frame, relative to the Orbit frame, in principle means that the satellite does one full rotation around the $-y_o$ axis during one orbit-period.

We start off by examining the hypothetical case where we are able to create torques about arbitrary axes even though this is not possible in reality as the magnetic coils can only create a torque perpendicular to their magnetic field. A possible controller could be one that creates a torque in the opposite direction of the current rotation, such as:

$$\boldsymbol{\tau}_d^B = -d\boldsymbol{\omega}_{OB}^B \quad d > 0. \tag{4.14}$$

The subscript $_d$ is used to denote the desired torque vector. In order to assert system stability with this controller, we go back to our Lyapunov function in 4.8 or more precisely its derivative from 4.10:

$$\dot{V} = (\boldsymbol{\omega}_{OB}^B)^T \boldsymbol{\tau}_m^B \tag{4.15}$$

which, after inserting for the controller torque becomes:

$$\dot{V} = -d \cdot (\boldsymbol{\omega}_{OB}^B)^T \boldsymbol{\omega}_{OB}^B < 0 \quad \forall\, \boldsymbol{\omega}_{OB}^B \neq 0. \tag{4.16}$$

According to Theorem 2 in Section 2.4, it is easy to see from this equation that we truly do have a dissipative controller which constantly drains the satellite of its kinetic energy. There is a possibility that the satellite settles in an unstable equilibrium, this however can not last due to the constant presence of small disturbances, which in practice guarantee that the satellite reaches one of the four stable orientations from 4.12. The immediate problem with this controller though, is that it is completely hypothetical. This is because the coils on the satellite are fixed, and can only provide a momentum $\mathbf{m}^B$ that has to be mathematically crossed with the geomagnetic field $\mathbf{B}^B$ to produce a torque $\boldsymbol{\tau}_m$, as:

$$\boldsymbol{\tau}_m^B = \mathbf{m}^B \times \mathbf{B}^B. \tag{4.17}$$

This torque is always in the plane perpendicular to the $\mathbf{B}$ vector and is depicted in Figure 4.1. In the figure we depict $\boldsymbol{\tau}_m$ as a projection of $\boldsymbol{\tau}_d$ into the plane perpendicular to $\mathbf{B}$. Torque $\boldsymbol{\tau}_m$ is the torque closest to $\boldsymbol{\tau}_d$, which we are able to generate. We thus need to find the moment $\mathbf{m}$ which gives us torque $\boldsymbol{\tau}_m$ when crossed with $\mathbf{B}$ gives, as in 4.17.

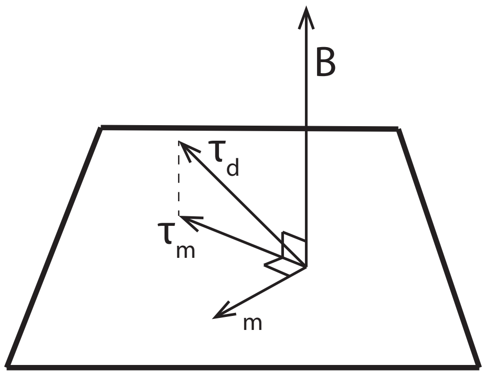


Figure 4.1: Projection of desired torque. Inspired by Figure 8.1 in [20].

From Figure 4.1 we can see that $\mathbf{m}$ can be expressed as:

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

where $f(\cdot)$ is an unknown scalar function making sure we preserve length of the $\boldsymbol{\tau}_m$ vector. Inserting this into 4.17 we get:

$$\boldsymbol{\tau}_m^B = f(\cdot)\left(\boldsymbol{\tau}_d^B \times \mathbf{B}^B\right) \times \mathbf{B}^B. \tag{4.19}$$

If we denote the angle between $\mathbf{B}^B$ and $\boldsymbol{\tau}_d^B$ as α, we find the length of $\boldsymbol{\tau}_m^B$ as:

$$|\boldsymbol{\tau}_m^B| = |\mathbf{m}^B||\mathbf{B}^B| = f(\cdot)|\mathbf{B}^B||\boldsymbol{\tau}_d^B||\mathbf{B}^B|\sin(\alpha). \tag{4.20}$$

Exploiting that $|\boldsymbol{\tau}_m^B| = |\boldsymbol{\tau}_d^B|\sin(\alpha)$, it is easy to find $f(\cdot)$:

$$f(\cdot) = \frac{1}{|\mathbf{B}^B|^2}. \tag{4.21}$$

We can now insert the previously unknown scalar to find our control algorithm:

$$\mathbf{m}^B = -\frac{d}{|\mathbf{B}^B|^2}\mathbf{B}^B \times \boldsymbol{\omega}_{OB}^B \tag{4.22}$$

$$\boldsymbol{\tau}_m^B = -\frac{d}{|\mathbf{B}^B|^2}\left(\mathbf{B}^B \times \boldsymbol{\omega}_{OB}^B\right) \times \mathbf{B}^B. \tag{4.23}$$

This controller turns out to be uniformly asymptotically stable (UAS). For proof see [20].

4.2.2 B-dot Controller

Relative to the dissipative detumbling controller in the previous section, the B-dot controller does not require direct knowledge of the angular velocity of the satellite. It does however require a derivative of the geomagnetic field, B-dot. This is a common controller in literature and is given as:

$$\mathbf{m}^B = -d\dot{\mathbf{B}}^B \quad d > 0. \tag{4.24}$$

This can be expanded as:

$$\begin{aligned}
\mathbf{m}^B &= -k\frac{d}{dt}\left(\mathbf{R}_O^B\mathbf{B}^O\right) = -d\left(\dot{\mathbf{R}}_O^B\mathbf{B}^O + \mathbf{R}_O^B\dot{\mathbf{B}}^O\right) \\
&= -d\left(-\mathbf{S}\left(\boldsymbol{\omega}_{OB}^B\right)\mathbf{R}_B^O\mathbf{B}^O + \mathbf{R}_O^B\dot{\mathbf{B}}^O\right) \\
&= -d\left(\mathbf{B}^B \times \boldsymbol{\omega}_{OB}^B + \mathbf{R}_O^B\dot{\mathbf{B}}^O\right).
\end{aligned} \tag{4.25}$$

From 4.25 we see that the first term is the same as the dissipative detumbling controller. From 4.24 we can see that this controller attempts to align the satellite with the geomagnetic field. Penalizing any change in the geomagnetic field means that this controller in practice would not rest until the satellite follows the the geomagnetic field exactly. This works well for a dipole model where the magnetic field has a simple structure and the field derivative is easily computable. For an IGRF model on the other hand it would mean constant draining of the battery and high power consumption on derivative computation. We present this controller because of its popularity, and because it is included in the code attached in the appendices. Our prototype controller implementation will only include the dissipative controller. For more information on this controller and its stability see [20].

4.3 Reference Controller for Large Deviations

Ultimately the attitude we wish to obtain is the one where the camera (payload), of the satellite points towards the earth. The camera will be mounted on one of the $10 \times 10cm$ sides. This face of the satellite is the one that the z_b-axis goes through and by definition in Section 2.1.5 is the axis with the lowest moment of inertia. We can therefore define our control goal as aligning the orbit and body frames. In other words reaching $\mathbf{R}_O^B = \mathbf{1}$, where $\mathbf{1}$ is a 3×3 identity matrix.

By using the Euler parameter vector as a measure of error we can extend the Lyapunov function in 4.8 as:

$$\begin{aligned} V = &\frac{1}{2}(\boldsymbol{\omega}_{OB}^B)^T \mathbf{I} \boldsymbol{\omega}_{OB}^B + \frac{3}{2}\omega_o^2 \mathbf{c}_3^T \mathbf{I} \mathbf{c_3} - \frac{1}{2}\omega_o^2 \mathbf{c}_2^T \mathbf{I} \mathbf{c}_2 + \frac{1}{2}\omega_o^2 (I_y - 3I_z) \\ &+ k\left[\boldsymbol{\epsilon}^T \boldsymbol{\epsilon} + (1-\eta)^2\right]. \end{aligned} \tag{4.26}$$

Where $k > 0$, and which at correct attitude has $\boldsymbol{\epsilon} = \mathbf{0}$ and $\eta = 1$, such that V is zero in this state and positive otherwise. Exploiting that $\boldsymbol{\epsilon}^T \boldsymbol{\epsilon} + \eta^2 = 1$, we are able to rewrite the last term as $2k(1-\eta)$. By taking the time derivative of V, the last term becomes $-2k\dot{\eta}$, which according to 2.15, can be written as $k\boldsymbol{\eta}^T \boldsymbol{\omega}_{OB}^B$. Combining this result with $\dot{V}$ found in 4.10, we obtain:

$$\dot{V} = \left(\boldsymbol{\omega}_{OB}^B\right)\left[k\boldsymbol{\epsilon} + \boldsymbol{\tau}_m^B\right]. \tag{4.27}$$

Which, assuming we can generate any arbitrary torque, gives us the control law:

$$\boldsymbol{\tau}_m^B = -d\boldsymbol{\omega}_{OB}^B - k\boldsymbol{\epsilon} \quad d, k > 0. \tag{4.28}$$

With this controller the equilibrium state $\boldsymbol{\omega}_{OB}^B = \mathbf{0}$, $\boldsymbol{\epsilon} = \mathbf{0}$, $\eta = 1$ becomes uniformly asymptotically stable, though not globally. For more stability information see [20], who also provides a gain requirement:

$$k > 8\omega_o^2(I_y - I_z). \tag{4.29}$$

In order to find the control law for the practical case when the moments are generated by the coils, we follow the same projection procedure as in 4.2.1 to arrive at:

$$\mathbf{m}^B = -\frac{d}{|\mathbf{B}^B|^2}\mathbf{B}^B \times \boldsymbol{\omega}_{OB}^B - \frac{k}{|\mathbf{B}^B|^2}\mathbf{B}^B \times \boldsymbol{\epsilon} \tag{4.30}$$

generating the torque:

$$\boldsymbol{\tau}_m^B = -\frac{d}{|\mathbf{B}^B|^2}\left(\mathbf{B}^B \times \boldsymbol{\omega}_{OB}^B\right) \times \mathbf{B}^B - \frac{k}{|\mathbf{B}^B|^2}\left(\mathbf{B}^B \times \boldsymbol{\epsilon}\right) \times \mathbf{B}^B \tag{4.31}$$

We note that the first part is identical to the dissipative detumbling controller. This can be exploited in the C-code to save memory and computational power, making the implementation more efficient.

4.4 Pulse Width Modulation (PWM)

Inside the satellite we will have access to voltage of a nearly[2] constant amplitude. This means that we cannot *directly* control the amplitude of the voltage and thus the current applied to the coils. Effectively this means that without further care we end up combining continuous time controller with a discrete time output. There is a solution however; and it comes in the form of Pulse Width Modulation (PWM).

PWM is similar to a Digital to Analog Converter (DAC). We choose to describe one of PWM's modes of operation through Figure 4.2, generated in Matlab. A microcontroller with a PWM contains counters which can be turned on to count, e.g. upwards, as illustrated by the thin seesaw in the upper graph. We can then pick a desired output, in our case voltage amplitude, which is stored into compare-registers on the microcontroller. From this we can configure the PWM to set one of its pins high[3] whenever the counter value exceeds value of the compare register. This is exactly what is

[2]Ignoring the fact that the battery loses some potential as it is drained and also varies with temperature changes.

[3]Microcontrollers can set their pins high (H) or low (L), enabling or disabling a voltage on the output, respectively.

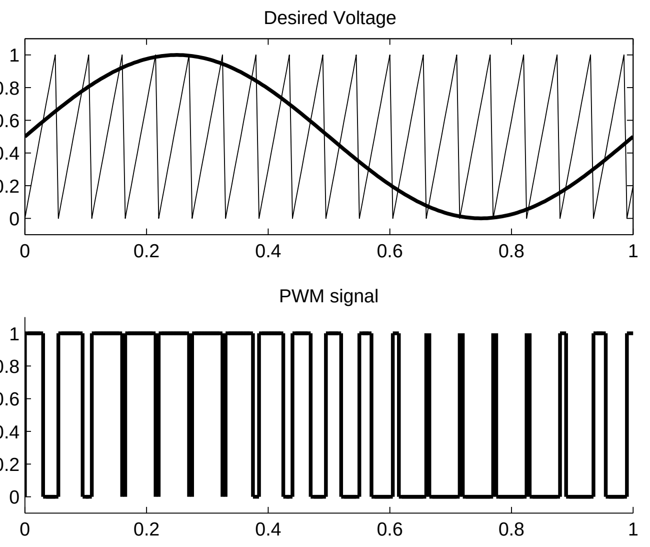


Figure 4.2: A sinusoid converted to a PWM signal.

done in Figure 4.2. The trick here is that if the period of the PWM is much smaller then the smallest time constant of the load, the fast changes in the PWM output will be blurred in the load. Thus the load will experience an averaged signal from the PWM which, in the case in Figure 4.2, is a sinusoid signal.

When using a magnetic coil as the load, it is desireable to have a small coil time constant, which can be found as:

$$\tau = \frac{L}{R} \tag{4.32}$$

where L is the inductance and R the resistance of the coil. Time constant τ can be described as $\tau = t - t_0$, where t_0 is the time when voltage is applied over the coil and t the time when the current through the coil reaches 64% of the final current. Thus, the smaller the time constant, the less additional dynamics are introduced to the system by the coil's charging and discharging time.

Chapter 5

Simulations

This chapter presents results of numerical simulations performed in Matlab. We start by simulating the two different geomagnetic field models, the dipole and the IGRF-11 model, in order to familiarize ourselves with these and to investigate model accuracy. Later we move on to the system as a whole. We use Matlab's $ode45$ function to numerically integrate the system with the state vector $\left\{x = \left[\begin{array}{ccccc} (\boldsymbol{\omega}_{IB}^{B})^{T} & \eta & \boldsymbol{\epsilon}^{T} & \lambda & p \end{array}\right]^{T} \in \Re^{1x9}\right\}$, where λ is the satellite's latitude and p its power consumption, $\int (\mathrm{Watt})\, dt$. The last state is not directly a physical state of the satellite and has no impact on satellite's orientation or its control; it is merely a simple and accurate way of integrating the power consumed by the satellite' control systems throughout the simulations. This state is, as mentioned, the integrated power usage of the system and relates to watt-hours as: $W \cdot h = \frac{p}{3600}$.

Picking different suitable initial states with multiple three dimensional frames for this satellite-earth system, was in our opinion tedious. We have therefore used the same standard as Soglo. The initial orientation and rotational velocity parameters are set as $\left[\begin{array}{cccc} \boldsymbol{\omega}_{OB}^{B} & \phi & \theta & \psi \end{array}\right]$, where $\boldsymbol{\omega}_{OB}^{B}$ is rotation of the body frame with respect to the orbit frame, expressed in the body frame coordinates, and $\left[\begin{array}{ccc} \phi & \theta & \psi \end{array}\right]$ are the Euler angles of a XYZ-rotation of the body frame relative to the orbit frame. These parameters are then, through Matlab function $eul2qua.m$ (see Appendix C) translated to $[\boldsymbol{\omega}_{IB}^{B}, \eta, \boldsymbol{\epsilon}^{T}]$ for an arbitrary initial latitude λ_0. Satellite's initial power consumption is always set to zero: $p_0 = 0$. All the files used in the simulations can be found in Appendix C.

5.1 Geomagnetic Model Comparison

Use of a dipole magnetic field model aboard the satellite would be beneficial through its simplicity. It is easy to extract the magnetic field values from the model and its derivative computation is just as trivial. This obviously reduces computation time, which in turn reduces the major control time constant and the system's power consumption. In Figure 5.1 we compare the dipole and the IGRF-11 models for longitudes 0 and 90°, to see if the system really needs to use the IGRF-11 model or if the dipole model is sufficiently accurate. The figure shows that there is fair amount of irregularities in the geomagnetic field, which are completely ignored by the dipole model - the geomagnetic field is longitude dependant, as expected. It is thus immediately obvious that the dipole model does not provide sufficient information, so that IGRF11 computation should be virtually impossible before falling back to the simple dipole model. Besides the irregularities, the fact that the dipole model ignores the y_o-axis components of the magnetic field is worrying as it may be enough to bring the system to an ill conditioned state, as shown in Figure 3.3.

		North	East	Vertical	Total field
Lat=53°	Lon=0°				
	On-line	14676.5	-668.9	34826.3	37798.4
	Matlab	14470.3	-675.9	34844.0	37735.3
Lat=80°	Lon=169°				
	On-line	3639.5	356.1	44711.8	44861.1
	Matlab	3553.0	365.9	44463.1	44606.3
Lat=-40°	Lon=-55°				
	On-line	13954.7	-1521.6	-13935.7	19780.1
	Matlab	13906.1	-1493.8	-14026.3	19807.7

Table 5.1: Comparison between internet IGRF-11 field calculator from NOAA NGDC and the implemented IGRF-11 model. All values are given in nanoteslas.

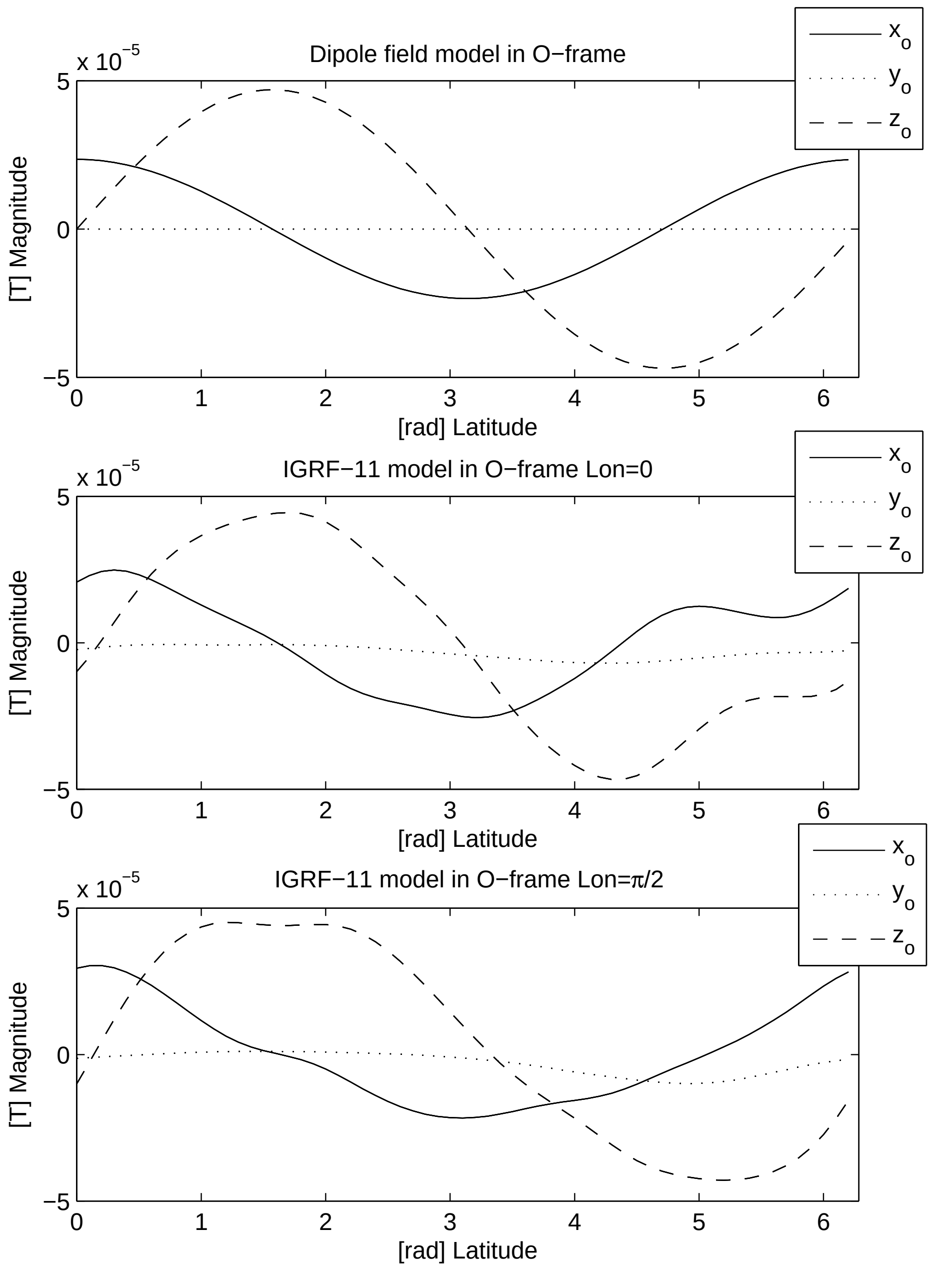


Figure 5.1: Dipole field and IGRF-11 model comparison for two different longitudes.

Errors	Lat=53° Lon=0°	Lat=80° Lon=169°	Lat=−40° Lon=−55°
Magnitude	0.17%	0.57%	0.14%
Direction	0.299°	0.086°	0.297°

Table 5.2: Direction and magnitude errors.

Now that we have ruled out the use of the dipole model we wish to check the accuracy of our IGRF model when compared with the most reliable field data which is obtained from the National Oceanic and Atmospheric Administration's (NOAA) National Geophysical Data Center's (NGDC) on-line calculator [17]. In Table 5.1 we compare the values of our simulated IGRF model and the on-line calculator for three different arbitrary positions.
The differences in values seem to be negligibly small. However, for small component values a small deviation may imply a large relative error, we therefore need to check the magnitude and directionality of the vectors to make sure we have a useable model. We compare the magnitude as a absolute percentage difference of the Matlab model compared to the on-line one. The angular deviation is found by using $\theta = \cos^{-1}\left(\frac{v_1 \bullet v_2}{|v_1||v_2|}\right)$, where v_1 and v_2 are the on-line and matlab found magnetic field values combined into vectors, assuming to have their origin in origo[1]. From these results in Table 5.2, we see that the error introduced by the our IGRF model really is negligible, and the model is well suited for our mission.

5.2 Detumbling Controller

Here we simulate the system behavior when using the energy dissipative controller to detumble the satellite. The moment from this controller, developed in 4.22, is:

$$\mathbf{m}^B = -\frac{d}{|\mathbf{B}^B|^2}\mathbf{B}^B \times \boldsymbol{\omega}_{OB}^B$$

generating the torque:

[1]E.g. $v_1 = \begin{bmatrix} 14676.5 & -668.9 & 34826.3 \end{bmatrix}$

$$\boldsymbol{\tau}_m^B = -\frac{d}{|\mathbf{B}^B|^2}\left(\mathbf{B}^B \times \boldsymbol{\omega}_{OB}^B\right) \times \mathbf{B}^B.$$

The main objective of this section being to provide the reader with some intuition of the satellite's dynamics during the detumbling phase, we pick a very simple initial condition, as shown in Table 5.3.

$\boldsymbol{\omega}_{OB}^B$	$=$	$\begin{bmatrix} 0 & -0.2 & 0 \end{bmatrix}^T$
η	$=$	1
$\boldsymbol{\epsilon}$	$=$	$\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$
λ	$=$	0
p	$=$	0
d	$=$	$4 \cdot 10^{-5}$
Satellite mass	$=$	2 kg
Simulation time	$\approx$	2896 sec ($\frac{1}{2}$orbit)

Table 5.3: Initial state and simulation parameters for the dissipative detumbling controller.

This gives us the following initial state vector:

$$\begin{aligned} x_0 &= \begin{bmatrix} (\boldsymbol{\omega}_{IB}^B)^T & \eta & \boldsymbol{\epsilon}^T & \lambda & p \end{bmatrix}^T && (5.1)\\ &= \begin{bmatrix} 0 & (-0.2-\omega_o) & 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.^T && (5.2) \end{aligned}$$

Putting it in layman's terms, we start with the satellite's body frame perfectly aligned with the orbit frame at zero latitude and zero longitude[2]. The satellite starts out with $0.2\frac{\text{rad}}{\text{sec}}$ angular velocity about the negative y_b axis, relative to the orbit frame. Note that the simulations were done while the maximum satellite weight was still uncertain and thus set to 2kg. Figure 5.2 shows

[2]Longitude is not a part of the initial condition because this parameter does not change - in simulations we assume that we have a perfectly polar orbit.

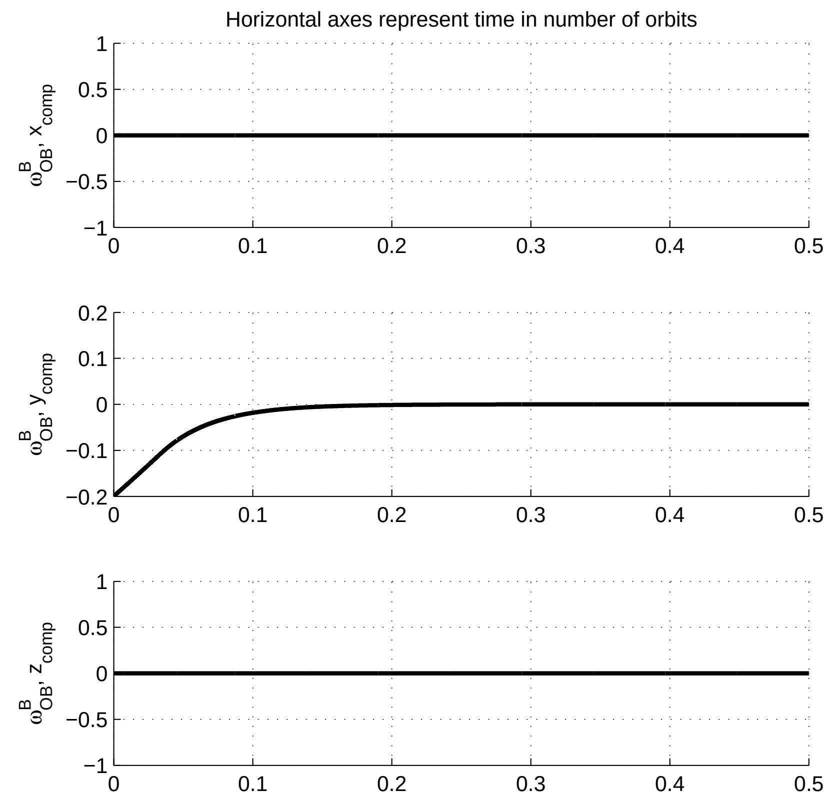


Figure 5.2: Satellite rotation, $\boldsymbol{\omega}^B_{OB}$ with the detumbling controller engaged.

the values we primarily are interested in, namely the satellite's rotation with respect to the orbit frame - this is the energy the controller is attempting to dissipate. We observe how the detumbling controller elegantly dissipates the energy from the y_b axis without tempering with the two other axis. These remain stable because the satellite is perfectly oriented with respect to the orbit frame.

Figure 5.3 shows the Figure A.3 when we zoom in on the interesting dynamics. We immediately notice that the moments from the z^+ and x^+ faces seem identical besides a small phase difference. This is perfectly reasonable,

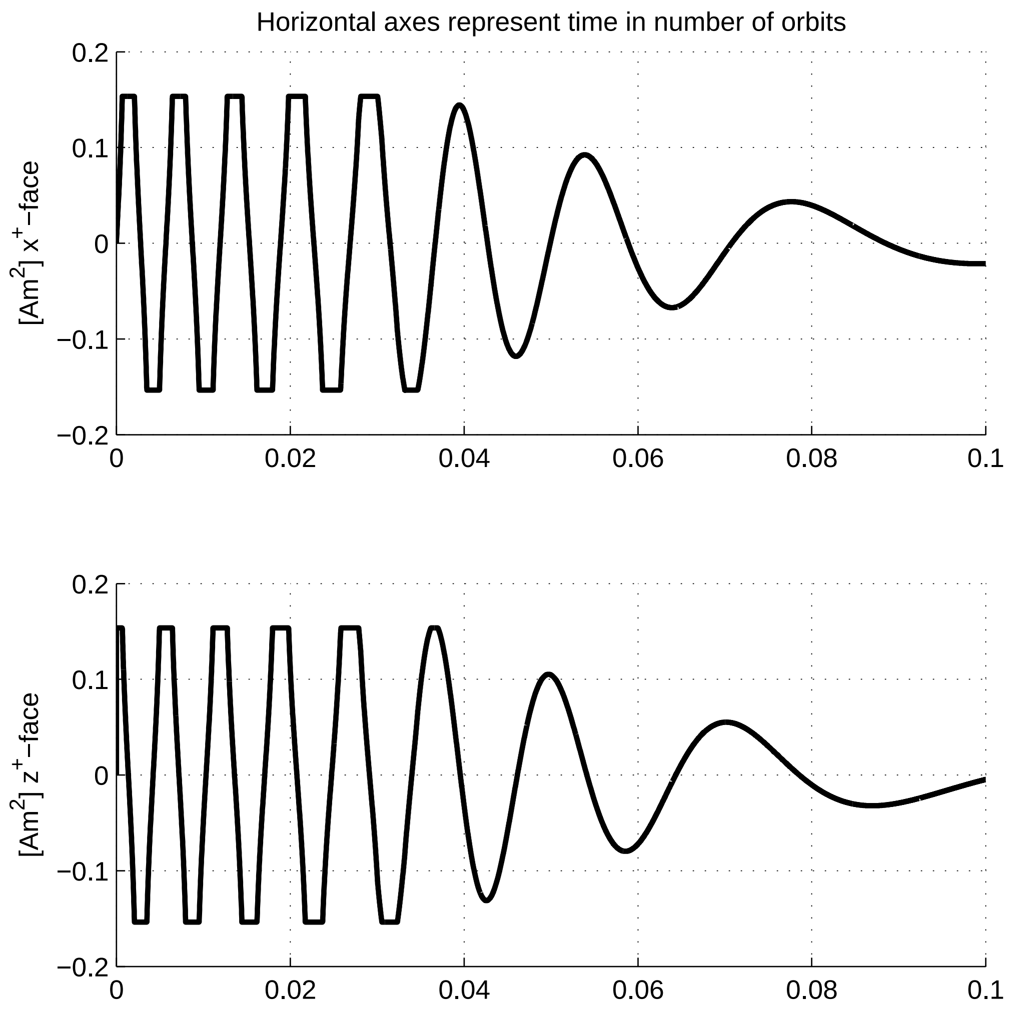


Figure 5.3: Moments generated by the controller. Enhancing interesting dynamics from Figure A.3.

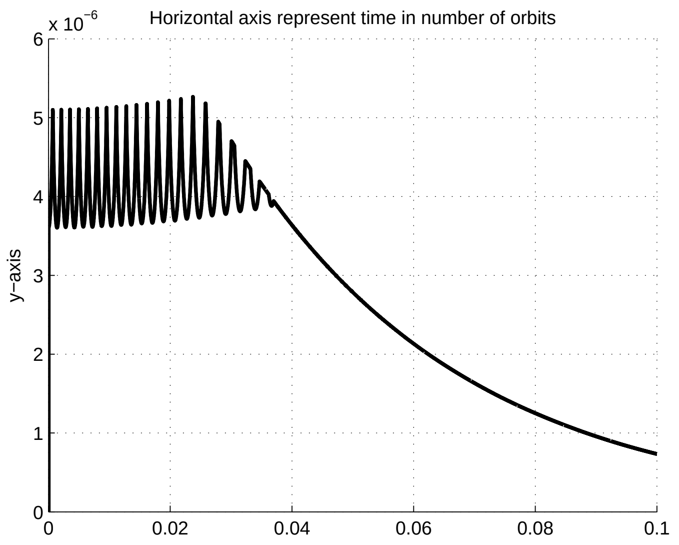


Figure 5.4: The generated torque. Enhancing interesting dynamics from Figure A.4.

expected, and is further explained in Appendix A. Besides their structure, we can see how that the moments are saturated when the electrical current of the desired, optimal moments exceeds the plausible current, available for our given voltage.
The moments from Figure 5.4 all create a torque about the y axis to detumble the satellite; this torque is shown in A.4 with the most interesting dynamics enhanced here in Figure 5.4. The almost comb-filter-like structure appears because of the relatively small controller gain - with a larger gain we would only have a straight line as the currents through the coils would constantly be saturated.

Lastly we examine the systems power consumption properties. In Figure 5.5 we are able see the total power consumption of the coils. We observe that the system uses just under 55 Joules $= 0.0153\frac{Watt}{Hours}$ to dissipate the kinetic energy from the satellite. The energy dissipation can be seen in Figure A.2. The electrical energy consumed by the coils is rather low, especially when taking into account that the detumbling phase is a one-time scenario. We have to remember though, that these simulations were performed while the

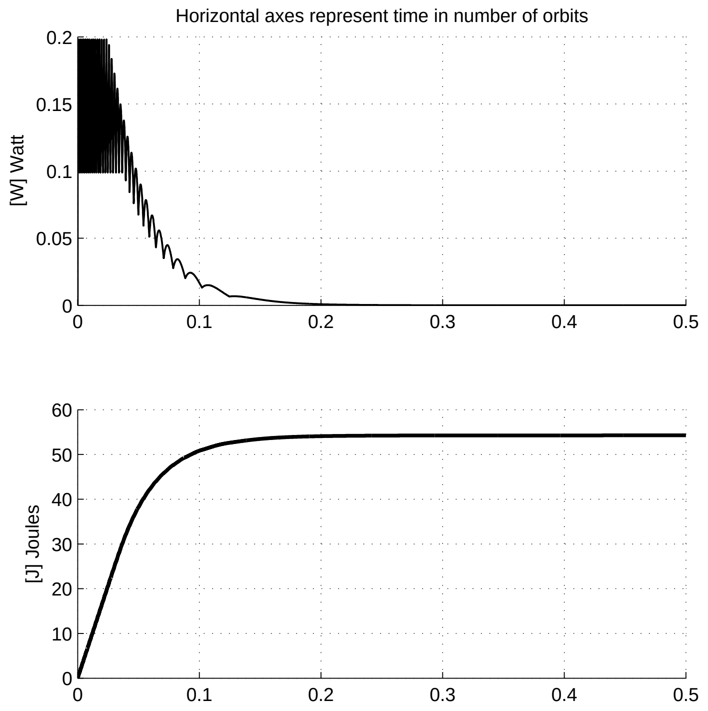


Figure 5.5: Controller's power consumption.

satellite's weight was assumed to be 2kg.

It is important to note that it is impossible to use the gain of this controller to affect satellite's state after detumbling. The equilibrium state reached at the end of detumbling phase can be seen as chaotic (changing the initial parameters may result in a completely different stable equilibrium state). Increasing the gain, to a certain point[3], does however decrease the detumbling time at the cost of increasing power consumption.

[3]Before system starts to oscillate.

5.3 Reference Controller

Here we simulate the system behavior in its full complexity; starting with the detumbling and ending with reference control. Simulation presented here was done using a 10th order IGRF model. The algorithm for the coil moment for this controller developed in 4.30 is:

$$\mathbf{m}^B = -\frac{d}{|\mathbf{B}^B|^2}\mathbf{B}^B \times \boldsymbol{\omega}^B_{OB} - \frac{k}{|\mathbf{B}^B|^2}\mathbf{B}^B \times \boldsymbol{\epsilon}$$

generating the torque:

$$\boldsymbol{\tau}^B_m = -\frac{d}{|\mathbf{B}^B|^2}\left(\mathbf{B}^B \times \boldsymbol{\omega}^B_{OB}\right) \times \mathbf{B}^B - \frac{k}{|\mathbf{B}^B|^2}\left(\mathbf{B}^B \times \boldsymbol{\epsilon}\right) \times \mathbf{B}^B.$$

The initial parameters for the simulations are shown in Table 5.4, where we chose the same initial tumbling as in the previous simulation of the detumbling controller. Initial Euler parameters are those corresponding to an XYZ-rotation of $\begin{bmatrix} 80^\circ & 50^\circ & 170^\circ \end{bmatrix}$ which was randomly selected in couple of tries, making sure the satellite does not reach $\mathbf{R}^B_O = \mathbf{1}$ using only the detumbling controller. The gain d is same as for the detumbling controller, while $k = 5 \cdot 10^{-8} > 8\omega_o^2(I_y - I_z) \approx 4.7 \cdot 10^{-8}$, which is the gain criterion from Section 4.3.

$$\begin{aligned}
\boldsymbol{\omega}^B_{OB} &= \begin{bmatrix} 0 & -0.2 & 0 \end{bmatrix}^T \\
\eta &\approx -0.210 \\
\boldsymbol{\epsilon} &\approx \begin{bmatrix} 0.373 & -0.552 & 0.715 \end{bmatrix}^T \\
\lambda &= 0 \\
p &= 0 \\
d &= 4 \cdot 10^{-5} \\
k &= 5 \cdot 10^{-8} \\
\text{Satellite mass} &= 2\ \text{kg} \\
\text{Simulation time} &\approx 23170\ \text{sec (4 orbits)}
\end{aligned}$$

Table 5.4: Initial state and simulation parameters for the reference controller.

For the first 1.5 orbits, the satellite is controlled by the detumbling controller, dissipating its kinetic energy. After this period the reference controller takes over; it is these dynamics we are interested in here. In Figure 5.7 we depict the Euler angles for the whole simulation, which are extracted from the quaternions in Figure A.5. One might think from the figure that the reference controller drives the satellite to a wrong attitude, as Euler angles $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$ correspond to quaternions $\eta = 1$, $\boldsymbol{\epsilon} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$. It is not that trivial though; the conversion from quaternions to Euler angles does not produce unique angles[4], rather unique attitude. What we are trying to say, is that the Euler angles in Figure 5.7 represent the same attitude as angles $\begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T$ do - rotating the satellite 180° about each of the three principal axes brings the satellite back to its original position. We can also notice minor yaw-axis oscillations towards the end of the simulation. These oscillations decay but can also be reduced by allowing the detumbling controller to slow the satellite even further, because, as we can see from Figure A.6, the satellite still has a small angular velocity $\boldsymbol{\omega}_{OB}^{B}$ after the initial 1.5 orbits.

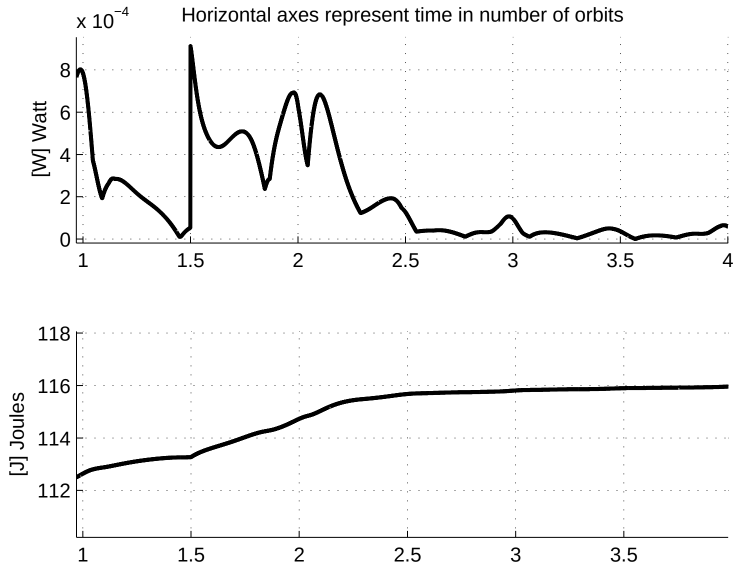


Figure 5.6: Power consumption for the full simulation.

[4] $+180° = -180°$.

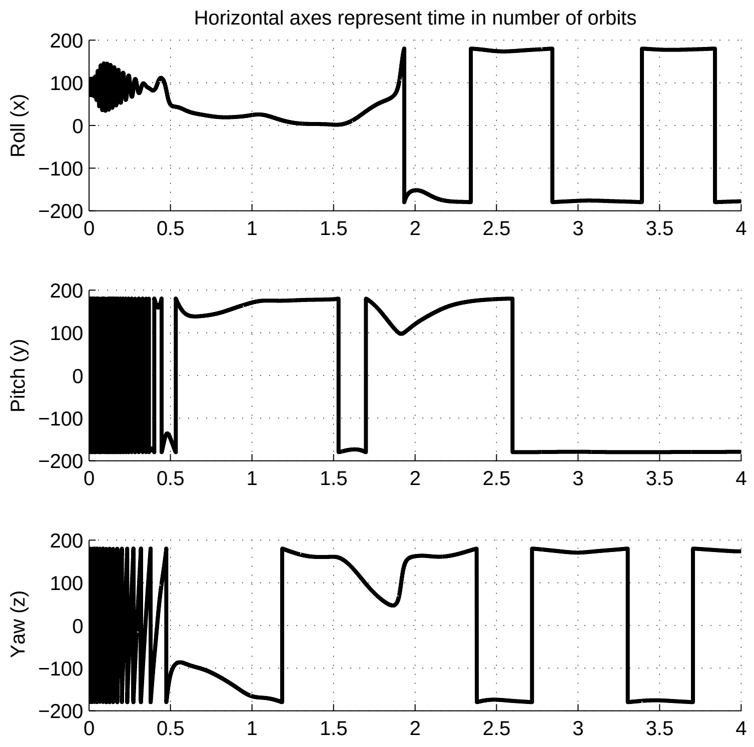


Figure 5.7: Euler angles for the full simulation.

Figure 5.6 shows the power consumption during the reference control. We can see that the satellite receives a "kick" when this controller is engaged. The total energy consumed by the reference controller amounts to 2J, barely having an effect on the battery. In Figure A.7 we can see the moments, which generate the torques in Figure A.8 amounting to this power consumption.

Chapter 6

Prototype Design

This chapter explains our approach and reasoning while designing the first prototype for the attitude control system. First we start off by identifying the hardware components necessary to build a fully functional control system. Secondly we design a circuit schematic for the connections of the components in the system. Thirdly we design a Printed Circuit Board (PCB) and mount all the necessary components. Finally we put all of the components together and program the microcontroller.

Throughout this whole chapter we try to point out the pitfalls when designing such a system in order to make this process as simple as possible for anyone attempting to recreate our results.

6.1 Hardware

Laid out in simple terms, we wish to run our control software on a microcontroller which then runs a current through a coil, creating a magnetic field used for control. We immediately realize that if we are to retain any controllability, the current applied to the coil has to be able to flow in both directions; forward and reverse. An H-bridge is a relatively simple electronic circuit designed for such a task.

Note that the attitude control module is a subsystem of the attitude determination and control system (ADCS). Attitude determination module was being designed in parallel to the control module; thus some component choices are *implicit, in order to create a fully functional ADCS.*

Microcontroller choice

The satellite attitude control algorithms implemented are not very computationally demanding. The only slightly intensive task is the magnetic field estimation from the IGRF-11 model, which changes slowly relative to its computation time. Therefore the primary requirement for the microcontroller choice is to have a minimum of three Pulse Width Modulation (PWM) channels, to be the inputs to the three coils. However, because the attitude determination prototype design was ahead of the attitude control prototype, we chose to use the same controller as it met all the control module requirements. The microcontroller used is thus a 64-pin Atmel ATMEGA2561v, with the data sheet [2].

H-bridge choice

Our choice of H-bridge was mainly based on its operating voltage ranges and its simplicity. Logic Supply Voltage (V_{CC}) for the H-bridge is determined to be 3.3V and its Load Supply Voltage (V_{BB}) to be 5V, by the power management design group. We wish to keep the system as simple as possible to avoid unnecessary weaknesses. We selected the 16-pin Allegro Microsystems A3953SLB-T SOIC, H-bridge. From its data sheet in [1] we find its Logic Supply Voltage Range: 3 - 5.5V and its maximum Load Supply Voltage: 50V. These specifications meet our requirements. In Figure 6.1 we provide a simple illustration of a H-bridge functionality, which also explains its name. If we define forward motor operation as “opening” transistors T1 and T4 while “closing” T2 and T3, it is easy to see that reverse operation can be obtained by opening transistors T2 and T3 while closing T1 and T4. This opening and closing is done through the H-bridge’s PHASE pin, explained in Section 6.2. All of this is thoroughly described in the data sheet. For reference we also present the physical layout of the component in Figure 6.2.

Unfortunately we have to mention that soon after ordering the H bridge we were made aware that it was deemed Pre-End of Life, meaning that the product was approaching end of life. This is a bit unfortunate for anyone trying to recreate the exact results presented in this thesis. However the impact of a different H-bridge should be minimal.

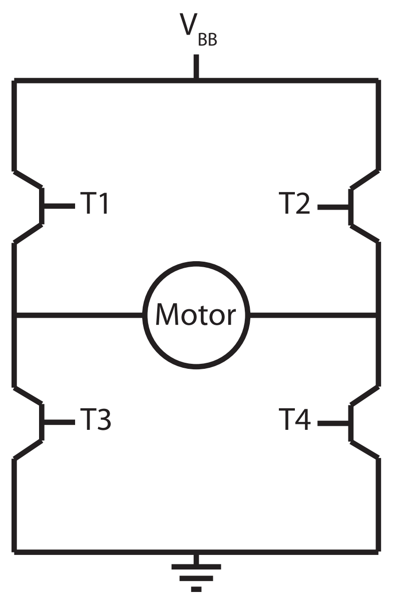


Figure 6.1: Basic H bridge structure.

Magnetic Coil

Three main factors primarily need to be considered when designing a coil for an application of this sort. We assume a 5V power supply.

1. Power
2. Size
3. Weight

We based our design on the coil's[1] maximum power consumption. We do not want to short-circuit the battery by having a negligible resistance in the coil, nor do we want to have an effect barely producing a magnetic field. As very few satellite specs were worked out at this point we started with an assumption that a $\frac{1}{4}$Watt maximum power consumption by the coil is reasonable. Using Ohms law:

$$V_z = R_z \cdot I_z \tag{6.1}$$

[1]We designed the coil for the 10×10cm, z^+-face only. The design procedure for the other two is nearly identical, with only one parameter (length) being different.

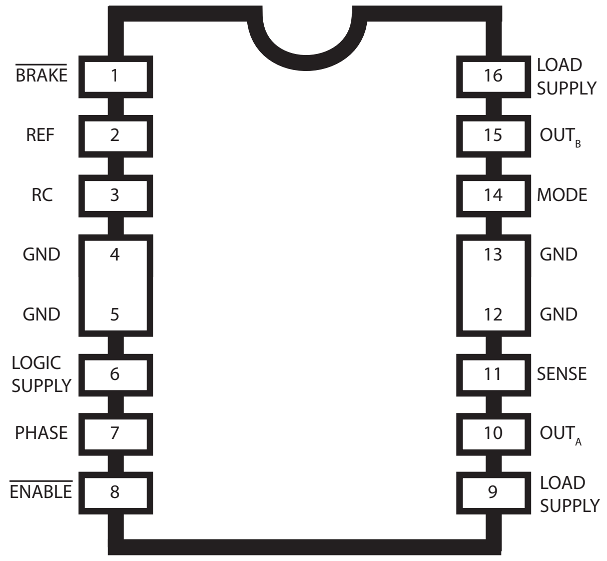


Figure 6.2: H-bridge physical layout.

where V is voltage, R resistance and I current, together with power law:

$$P = V_z \cdot I_z \tag{6.2}$$

where P is power, we were able to directly extract the coil's resistance that meets this requirement:

$$\frac{1}{4} = V_z \cdot I_z = \frac{V_z^2}{R_z} = \frac{5^2}{R_z} \rightarrow R_z = 100\Omega. \tag{6.3}$$

Because conductor resistance is circumference dependent, this problem does not have a unique[2] solution. The list of available wire diameters (circumferences) can be found in the American Wire Gauge (AWG) standard. The AWG standard can be found in many booklets and on the web, e.g. [18]. The most interesting values are for reference included in B.1, where our final conductor choice is highlighted. In order to have space for the coil inside the satellite, we constrain its size to 80×80mm. We plot, in Figure 6.3 the number of turns, maximum momentum and approximate[3] weight for differ-

[2]We have two variables: number of turns and wire circumference but only coil's maximum power consumption as a constraining equation.

[3]We find the weight of a coil with all turns the same size and without insulation - this is an approximation in our case. Putting much effort into exact calculations at this stage

ent conductor circumferences. We explain how these parameters were found by using wire diameter of 0.227mm as an example; the same conductor diameter as in our final prototype. This diameter was selected in order to obtain the largest possible maximum momentum while retaining a feasible weight of 0.084Kg.

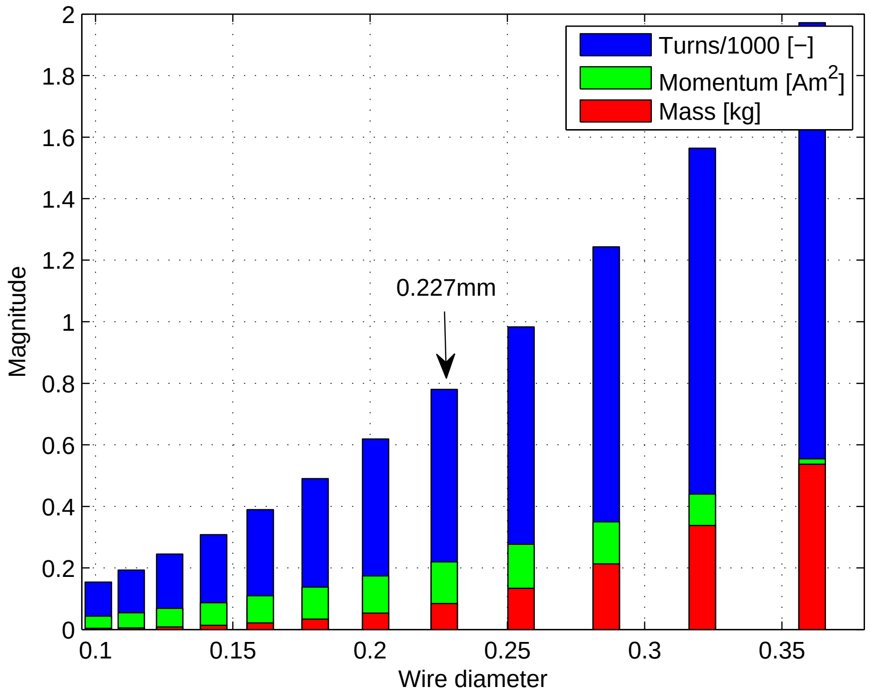


Figure 6.3: Turns, maximum momentum and approximate weight for different conductor circumferences.

Number of turns for the coil is identified as the quotient of the following division:

$$\text{Floor}\left\{\frac{(100\Omega)}{(0.427\frac{\Omega}{\text{m}})\cdot[2(0.075+0.075)]\text{m}}\right\}=780 \tag{6.4}$$

is futile because none of the exact inner satellite dimensions are known.

where $0.427\frac{\Omega}{m}$ is extracted from B.1 and sides of 75mm are selected in order to ensure that size of the coil stays within 80×80mm. Inserting resistance, voltage and the number of turns into 3.26 we find the maximum magnetic moment to be:

$$780 \cdot 0.075^2 m^2 \cdot \frac{5\mathrm{V}}{100\Omega} = 0.219\mathrm{Am}^2. \tag{6.5}$$

We can also find the approximate weight of the coil:

$$780 \cdot 2(0.075 + 0.075)\mathrm{m} \cdot 0.3577 \cdot 10^{-3}\frac{\mathrm{Kg}}{\mathrm{m}} \approx 0.084\mathrm{Kg}. \tag{6.6}$$

An illustration with the parameters for the final coil design can be seen in Figure 6.4. We calculate the coil's inductance using the formula for a square loop coil from [22]:

$$\begin{aligned} L &\approx \frac{2N^2\mu_0 l}{\pi}\left[\sinh^{-1}\left(\frac{l}{w}\right) - 1\right] \\ &= \frac{2 \cdot 780^2 \cdot 4\pi \cdot 10^{-7} \cdot 0.075}{\pi}\left[\sinh^{-1}\left(\frac{0.075}{0.1135 \cdot 10^{-3}}\right) - 1\right] \approx 226\mathrm{mH} \end{aligned} \tag{6.7}$$

where $N = 780$ is the number of turns, $\mu_0 = 4\pi \cdot 10^{-7}$ is the vacuum permeability, l the (average) length of coil's sides and w the wire radius. From this we can also find the coil's time constant:

$$\tau = \frac{L}{R} = \frac{0.226}{100} = 2.26\mathrm{ms}. \tag{6.8}$$

Additional components

The overall system requires a set of standard components to enable safe and stable operation, in addition to the integrated circuits (ICs) mentioned. These are summed up here, in Table 6.1 as part of the hardware section, while the reasoning behind these is presented in the next section.

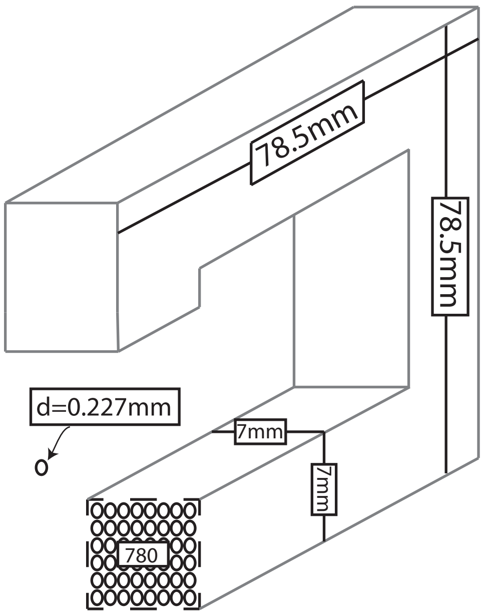


Figure 6.4: Illustration of the designed coil.

Capacitors	Resistors	Misc
4x 100nF	1x 10$k\Omega$	1x 16Mhz crystal oscillator
2x 12pF	1x 30$k\Omega$	18 connector pins
1x 470pF	1x 0.5Ω[4]	
1x 47μF		

Table 6.1: Additional components.

6.2 Circuit Schematic

The circuit schematic was created using the freeware version of EAGLE 5.11.0 from CadSoft. Attitude control module was appended to the already existing schematic for the attitude determination module, which included the microcontroller, sensor, their necessary capacitors and resistors, together with JTAG and power supply connectors. The microcontroller in the schematic is an ATMEGA128A, not the implemented ATMEGA2561V, due to late controller changes. The pin functionality of the two controllers was for our purposes identical and thus the original controller was retained in the schematic. The microcontroller together with components from Table 6.1 was selected from EAGLE component library, while the sensor and the H bridge were designed manually and then added to the library as well as the schematic. The dimensions of the H-bridge were extracted from the last page in its data sheet [1].

In Figure 6.5 we can see the final setup. We observe the microcontroller on the left (when seen from the side), H-bridge in the upper right and the sensor in the lower right corner. Three of the $100nF$ capacitors are installed to decouple the microcontroller's V_{CC} from ground while the last one decouples the reset pin. This is to done to stabilize the voltage supply. The external clock and its connection setup is described in the "System Clock and Clock Options" chapter in [2]. For sensor connection we refer to [10]. The H-bridge is connected according to its typical application setup, as shown in Figure 4 in the data sheet [1]. Note that this scheme together with the prototype in later sections, includes hardware for a single coil. This is done due to economic reasons, and as nearly all parts of the control module can be tested using a single coil.

Before any further explanation on H-bridge connection, we need to look at its truth table[5]. We set $\overline{BRAKE}$ and MODE low (see [1] for better understanding) and present the relevant truth table data[6] in Table 6.2. From this table we see that PHASE defines the direction of the current by deciding which of the outputs, OUT_A or OUT_B, is to be set high. $\overline{ENABLE}$ on the other hand overrides PHASE and decides whether there should be any

[5]Truth table of a H bridge relates its output values to its input conditions. H - high/on, L - low/off.

[6]We have modified the table to show actual inputs - $\overline{ENABLE}$ instead of ENABLE.

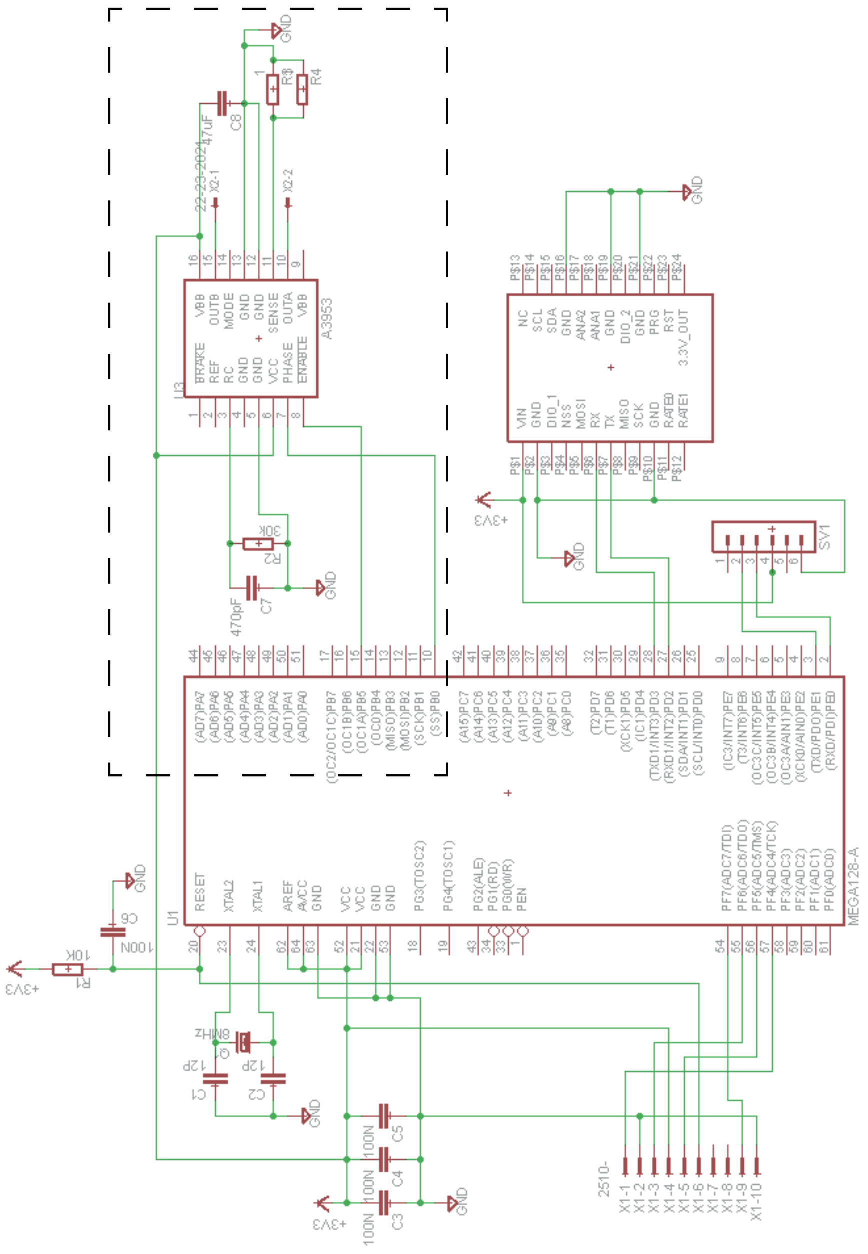


Figure 6.5: Circuit schematic.

$\overline{ENABLE}$	PHASE	OUT_A	OUT_B	DESCRIPTION
L	-	Off	Off	Standby
H	H	H	L	Forward, Slow Current-Decay Mode
H	L	L	H	Reverse, Slow Current-Decay Mode

Table 6.2: H bridge Truth Table from [1].

output at all.

In Figure 6.6 we enhance the dashed rectangle region from Figure 6.5 to comment further on how everything is brought together. Note that we use Timer 1 and its respective compare registers for the PWM signals.

Let us now continue by explaining the two connections between the microcontroller and the H-bridge. The fifth port B-pin of the microcontroller, (OC1A)PB5, is the output for the first Timer 1 compare register. This is trivially connected to the the $\overline{ENABLE}$, creating the (OC1A)PB5 $\longleftrightarrow$ $\overline{ENABLE}$ connection. The second connection is (SS)PB0 $\longleftrightarrow$ PHASE, where (SS)PB0 is an arbitrary[7] microcontroller pin used to set the direction of the current. Note lastly, besides the typical H-bridge application components, that we apply the same supply power to both Logic and Load inputs of the H-bridge. This is for prototype testing simplicity - supplying power to the system from a single source (USB). The two connector pins in the figure, X2-1 and X2-2, are pulled out for easier coil connection.

6.3 Print Circuit Board (PCB)

Once the schematic design was completed we moved on to EAGLE's board view in order to design the physical layout of our PCB. Circuit connections are created in the circuit schematic as mentioned, but it is here (board view) that the designer creates the physical paths of the conductors and places all the components on the board. The designer creates the actual physical

[7]We avoid using PB6 and PB7 as these are suitable as $\overline{ENABLE}$ signals for the last two coils

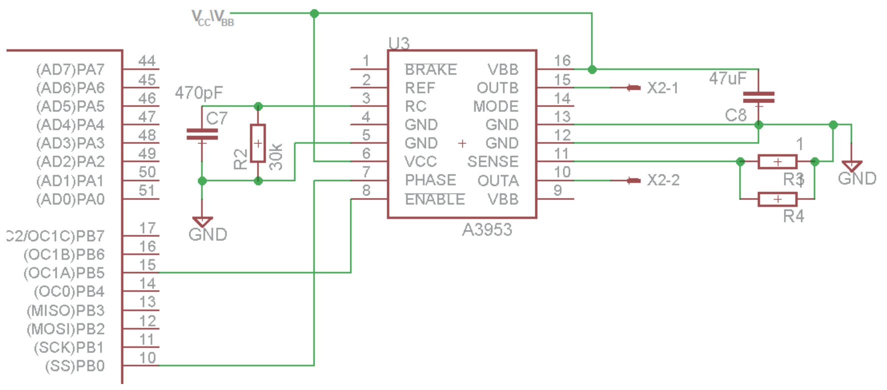


Figure 6.6: Control module enhanced.

setup/look of the PCB. For the prototype we decided to use a two layer PCB, which apparently gives us more then enough space to place all of the required components and their paths. Figure 6.7 depicts the final board design for the first ADCS prototype, with only one control coil. From the figure we see that this board measures approximately 87×50mm. Although the board is lackin two control modules, judging from the fact that there is still plenty of space available on the current size board and that the final board dimensions are $\backsim 90 \times 90$mm, space on the ADCS module does not seem to be an issue.

Practicalities

As this was our first time designing and printing a PCB, we will try to shed light on the problems we encountered so that possible pitfalls may be avoided in future design and research. Some suggestions given here are not followed in our design. This was because we learned during the process, and did not feel the need to create a whole new prototype just to make minor changes and enforce slightly safer operation when already operating in a controlled environment.

- We urge the designer to create short paths from the unused component pins on the board, and end them in a *via*[8]. This makes slight modifications of the prototype much simpler since one does not need to begin anew if a slight change is required; one can only add a wire to create

[8]Vias are small vertical electrical connections between different layers of conductors in a PCB.

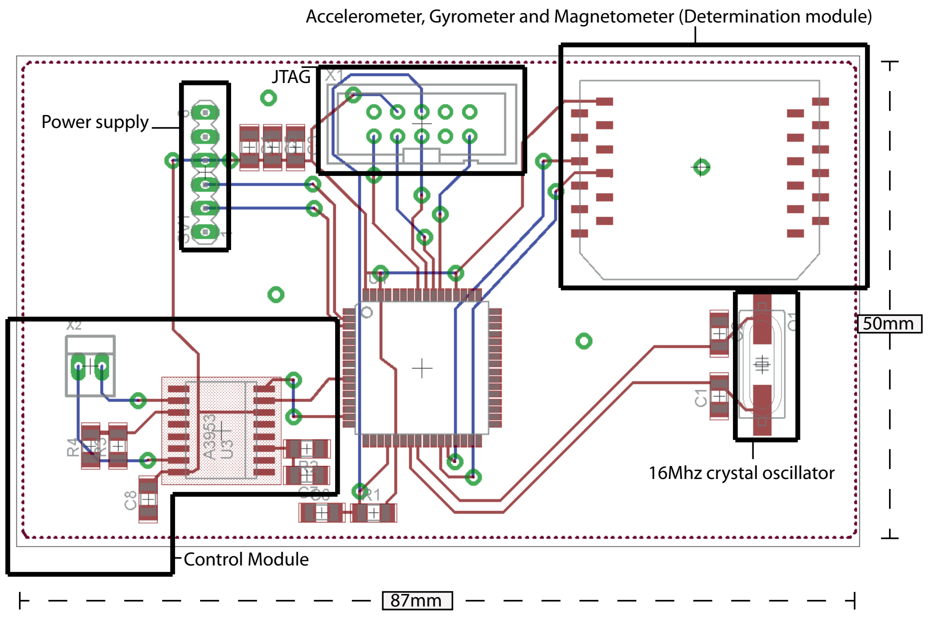


Figure 6.7: Board schematic.

the needed connection.

- Conductors connecting the microcontroller to the external clock in our schematic are excessively long. They should be made shorter whenever possible to reduce additive noise to the clock ticks.
- We placed three 100nF capacitors at the top of our board, between the power supply and the JTAG connectors. These should ideally be spread out all over the board to decouple V_{CC} on the microcontroller from the ground.
- When designing a PCB, connect connectors on the back side only. This is because soldering has to be done on one side of the board and it is unusual to place vias in the connector holes.
- After printing a PCB, pay attention when inserting vias, as they should *not* be applied to connector holes, unless an extra large drill bit is used for these holes.
- When soldering, solder the non-ground connection of a component first. It is much easier to start with this end as it is separated from the rest of

the board. Ground on the other hand is not[9], making heat conduction an issue for an inexperienced solderer.

6.4 Final Prototype Hardware

The prototype from the design in Figure 6.7, was printed using a LPKF ProtoMat S62 circuit board plotter, available at the Engineering Cybernetics lab located in the basement of the D-block, Elektrobygget, Gløshaugen. A non-conductive substrate board covered in copper sheets, is placed inside the circuit board plotter. Design schematics for the prototype are imported into BoardMaster and CircuitCAM software on the computer controlling the whole printing process, as described in [13] available at the lab. From here on the printing process is largely automatic. After printing the circuit board, all the required circuit components were soldered by hand. The soldering was done using a thin tip soldering iron heated to $450\,C^{\circ}$, using solder paste. The prototype created, with labeled parts, can be seen in Figure 6.8. Note that there is a connector missing on the power supply connector array. This is because only a 5-pin connector was available at the time, and the last pin on the power supply is totally disconnected from the system. Connecting this pin would only lead to better mechanical stability, which was not an issue, and was therefore omitted.

The final coil prototype can be seen in Figure 6.9. Immediately from the figure we can see that the coil size does not meet the design specifications nor the system requirements, making it unsuitable for the satellite. This was a slip-up on our part when designing the coil. First we made a miscalculation, where we worked with the average instead of absolute dimensions of the coil. The second error, and a possible pitfall, was not including conductor insulation in our calculations. Insulation thickness, according to Oddvar Landrø at the Department of Electric Power Engineering, is about $\frac{1}{10}$ of the conductor diameter, and must thus be taken into account when designing a coil.

The coil was measured with a letter weight and was surprisingly found to be exactly $0.084kg$, identical to the theorethical value from our miscalculations. Its resistance was measured to be $\backsim 100\Omega$. Measuring its inductance proved

[9]In our case. This is however design dependent.

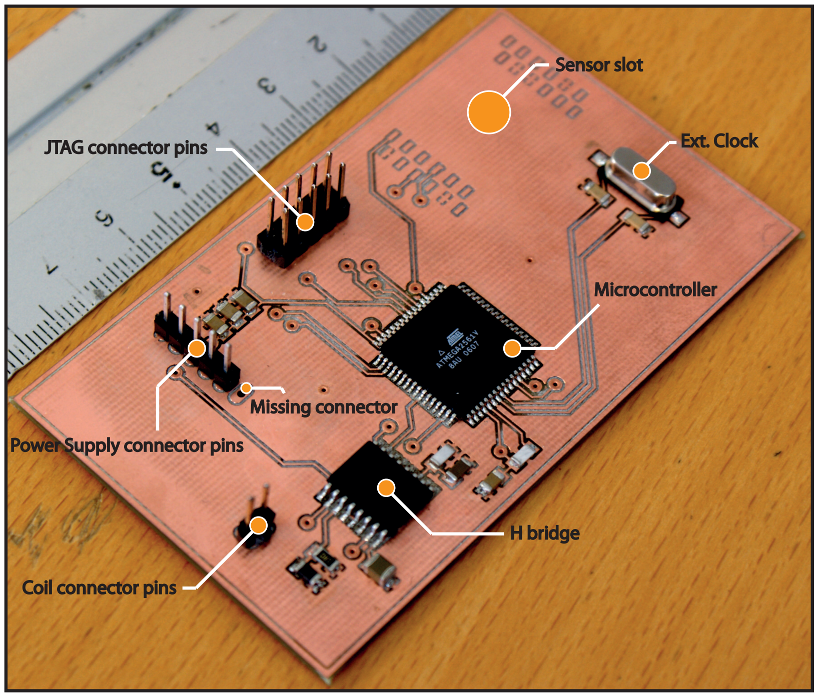


Figure 6.8: PCB prototype.

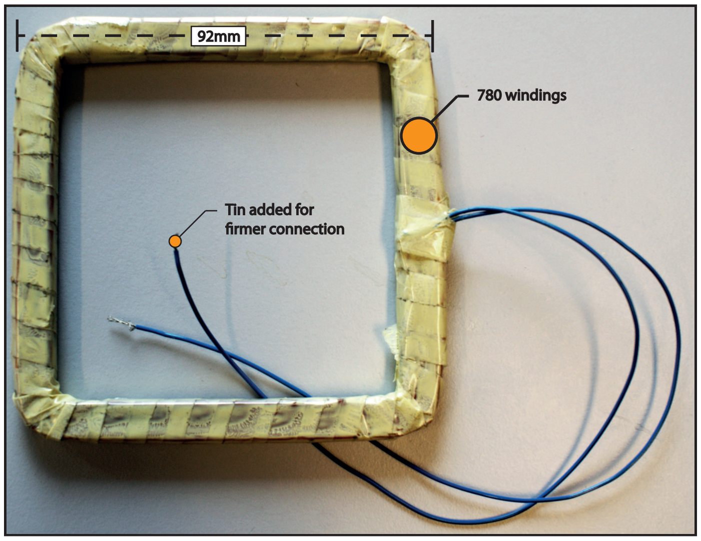


Figure 6.9: Coil prototype.

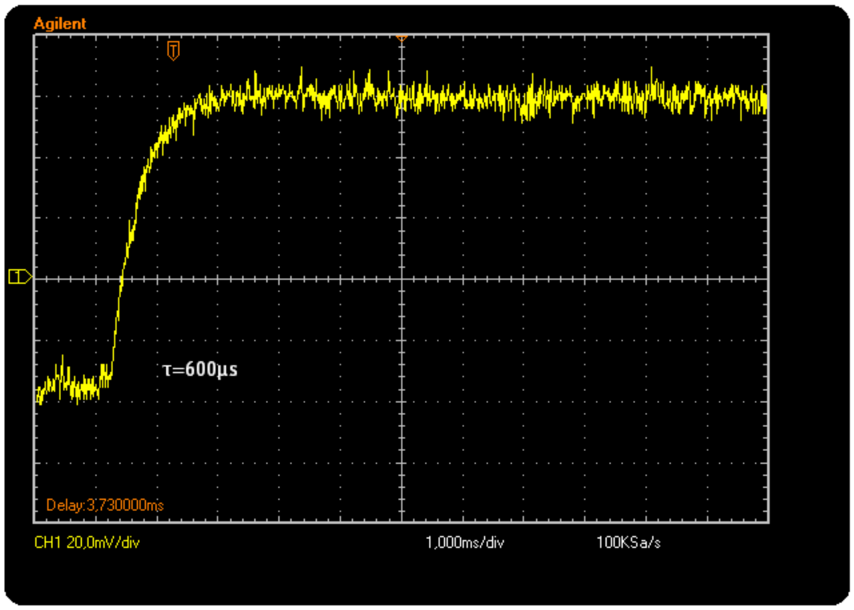


Figure 6.10: Measurement of coil's time constant.

to be a tougher task. We were unable to arrive at any consistant measurement using an analog multimeter and had to find the inductance through its time constant $\frac{L}{R}$. The measurement of the time constant was set up in the following manner: a square $1Hz$ wave from the signal generator, ISO-TECH GFG8219, was applied to the coil, while a galvanically separated oscilloscope was used to measure the current through the coil (more details on how the current is measured are available in Section 7.3). The time constant was measured to be $600\mu s$ which gives us an inductance of $600\mu s \cdot 100\Omega = 60mH$. This is approximately $\frac{1}{4}$ of the theoretical inductance. The primary reason for such a theoretical error is that the N^2 approximation in 6.7 assumes that all the turns of the coil occupy the physical space of a single turn. The oscilloscope graph of the coil's time constant measurement is depicted in Figure 6.10.

6.5 Programming the Microcontroller

The software for the prototype was programmed using C programming language, in Atmel's AVR Studio 4.18 integrated development environment

(IDE). The program itself was transferred to the microcontroller using an AVR Dragon Development board by Atmel, with the connection setup depicted in A.9, where the power supply for both cards was a 5V PC USB connection.

All of code written for the microcontroller is a direct translation of the Matlab-code, which was the first to be written. An important difference is the non-existence of matricies and their operations in C. Thus all the $n \times m$ matrices from Matlab are converted to $1 \times (n \cdot m)$ vectors in C. This requires special care when doing matrix operations; especially in the IGRF model, which is quite complex. All other computations, beside the IGRF model, were expanded by hand and hard-coded to reduce program complexity and avoid using any additional pseudo-matrix manipulating functions. The code for the microcontroller is included in the Appendix C as *AttitudeControl.c*. For easier reference, this code includes comments before all operations, presenting their Matlab counterparts.

For the PWM we use a 16-bit Timer1 which gives us high resolution and has three compare registers, meaning that one timer is enough to control all three coils. The initiation of the PWM is done in $PWM\_init()$ function:

```
void PWM_init(){
  // Waveform generation mode: PWM, Phase and Frequency Correct
    TCCR1A |= (0<<WGM11)|(0<<WGM10);
  TCCR1B |= (1<<WGM13)|(0<<WGM12);

  // No prescaling
  TCCR1B |= (0<CS12)|(0<<CS11)|(1<<CS10);

  // Clear on compare match when up-counting.
  // Set on compare match when down-counting.
  TCCR1A |= (1<<COM1A1)|(1<<COM1A0);
  TCCR1A |= (1<<COM1B1)|(1<<COM1B0);
  TCCR1A |= (1<<COM1C1)|(1<<COM1C0);

  // Resolution
  ICR1 = 0x0400 - 1;

  // DDRB |= (1<<DDB0)|(1<<DDB4);
  DDRB = 0xFF;
}
```

We identify all required register bits using the microcontroller's datasheet [2] as reference. First we choose to use a Phase and Frequency Correct PWM waveform generation, which is best explained in Figure 57 in the datasheet. The PWM timer counts up and down, having a triangle function as opposed to the seesaw in 4.2. Secondly, we choose not to use any prescaling on the clock in order to retain high resolution and frequency of the PWM. Setting this register starts the PWM timer. Thirdly, we choose to set the PWM output low on compare match when up-counting, and high on compare match when down-counting. This means that a compare register at 60% maximum value will produce a 60% duty cycle as shown in Figure 6.11. Here the square signal represents the PWM output, triangle the timer counting up and down, and straight line represents the compare register. Lastly we select the PWM's resolution and define the B-ports as out-ports. The ICR register tells the microcontroller how many steps it should make when counting up/down. By setting this register to e.g. 10 we tell the microcontroller to divide its up- and down-counting into 10 steps, meaning that we end up with bad resolution where the controller cannot distinguish compare register values c that lie in the set $\{(k < c < k + 0.1 \mid k = 0, 0.1, 0.2, ..., 0.9\}$. This would indicate that the optimal resolution is $0 \times FFFF$. However there is a downside; for a e.g. 10 step resolution, the Timer ticks 20 times altogether on one up-/ down-count. This means that a $16Mhz$ microcontroller can maximally generate a $800khz$ PWM signal:

$$\frac{CPU\ Speed}{2 \cdot Resolution} = \frac{16 \cdot 10^6}{20} = 800000. \tag{6.9}$$

Because the coil has an inductance, blocking a step change in current, and thus a step change in magnetic field, this frequency/resolution relationship has to be analyzed in order to have a coil that produces the desired magnetic field.

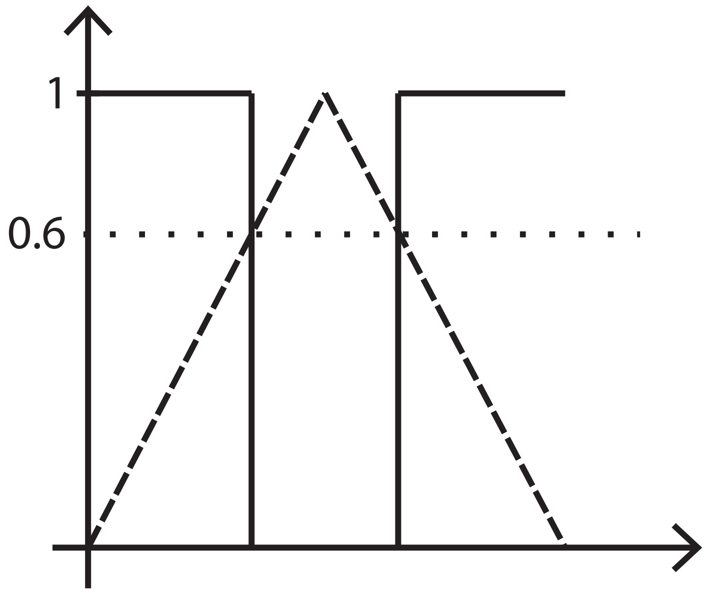


Figure 6.11: PWM compare register illustration.

Chapter 7

Prototype Simulations

7.1 Code Runtime

Here we analyze the runtime of the code implemented on the microcontroller. Looking at the code in *attitudeControl.c* in Appendix C, it makes sense to section the code in three parts: initialization done outside the main while loop, the while loop computations excluding IGRF model computation, and finally the IGRF model computation. We compute their runtimes by counting the number of clock cycle passing during each of the sections using one of the microcontroller's timers. Converting clock cycles into elapsed time is done by the following equation:

$$\frac{Cycles \cdot Prescaling}{CPU\, speed}. \tag{7.1}$$

While the CPU speed of the microcontroller is $16Mhz$ we prescale it by 64 to avoid overflow when counting clock cycles for IGRF computation.

Initialization of the code is completed in 104 clock cycles, which translates to $6.5\mu s$. The main loop computations are done in 4081 clock cycles, corresponding to $255\mu s$. IGRF computation time was analyzed for all 13 orders of the model which are presented in Table 7.1. There is no reason, at this point, to believe that its runtime, for any order, could deteriorate controller's integrity. However, processing power does drain the batteries, so we wish minimize the number of arithmetic operations. Due to time limitations we do not pursue this problem all the way through. We only provide the runtimes,

Order	1	2	3	4	5	6	7
Clock ticks (1/64 prescaling)	596	1504	2813	4521	6624	9134	12036
Time [ms]	2	6	11	18	26	36	48

Order	8	9	10	11	12	13
Clock ticks (1/64 prescaling)	15335	19036	23138	27637	32523	37808
Time [ms]	61	76	92	110	130	151

Table 7.1: IGRF runtime for orders 1 through 13.

so that in the future after some accuracy requirement analysis, conclusions may be drawn on what order model should be used.

7.2 Coil Temperature Dependence

The resistance of electrical components is temperature dependent. When it comes to the magnetic coil, especially, this dependence has to be mapped and adjusted for. This is because the coil's resistance is inversely proportional to its maximum current flow and thus its maximum magnetic moment. For this reason we perform an experiment where we vary the coil's temperature while measuring its resistance. We use a Vötsch Industrietechnik VT 4011 temperature test chamber for correct environment temperature manipulation.

Fourteen similarly[1] spaced measurements from $-40C^{\circ}$ to $+30C^{\circ}$ were taken. Plotting these in Matlab reveals a linear-looking relationship, which through first order polynomial (linear regression) yields coefficients $\{0.3668,\ 91.6063\}$ and the following approximating function:

$$R = 0.3668T \cdot T + 91.6063 \tag{7.2}$$

where R is the resistance and T the temperature of the coil. The measurements and the approximating function are depicted in 7.1 It is highly likely that the approximating function is even more accurate than it appears in the figure. This is because disturbances without a doubt did enter the system as we operated with limited time frames. In other words, had we waited

[1]Temperature test chamber had problems settling at desired non-negative temperatures. Therefore the jump at $0C^{\circ}$ and slight step variations.

longer at each temperature step for the temperature in the coil to stabilize, we could have obtained more accurate measurements.

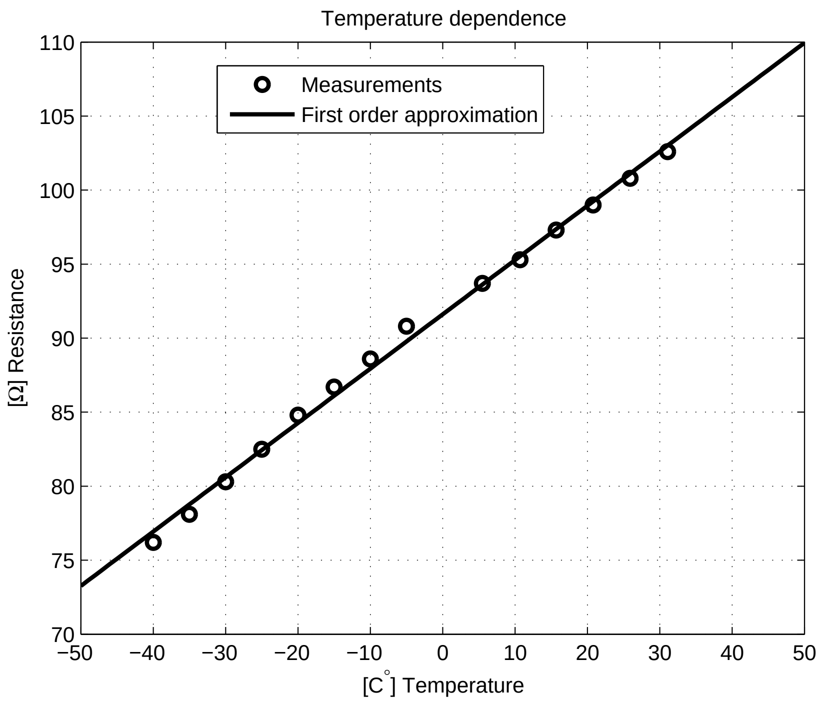


Figure 7.1: Temperature-resistance relationship for the prototype coil.

7.3 Coil Output

For final testing of the prototype the system was connected as in Figure A.10. The full setup consists of an oscilloscope needed to measure the signal from the PWM through the coil, and a separating transformer used for Galvanic separation (isolation) of the power supply form the PC and the oscilloscope. The Galvanic separator is required because switching the PWM's direction, while using an oscilloscope to measure voltage over the H-bridge's outputs $OUTA$ and $OUTB$, results in short-circuiting the whole system.

As it is rather easy to make a bad connection at one point or the other, we urge anyone attempting to recreate these results to work in a quiet environment, and if possible, to keep a finger on the H-bridge and the microcontroller when making a new connection while the system is running. This proved crucial in our case, because the system often creates a low, high frequency noise when a bad connection is made, and the components quickly start overheating. The user should in these cases immediately disconnect the power supply to the system or the newly added connection and rethink his or her approach.

All of the oscilloscope graphs were captured to PC using the 3000 Series Scope Connect Software v1.1.27 by Agilent Technologies found on the producers web page, together with the necessary drivers. Our first simulation was done to see if the prototype could output a correct *duty cycle*[2] and correct PWM frequency. In the C-code the duty cycle was set to 50% and resolution to 1024 bits. A 1024 bit resolution according to 6.9 should be:

$$\frac{16 \cdot 10^6}{2 \cdot 1024} \approx 7812.5Hz. \tag{7.3}$$

From the result in Figure 7.2 we see that the duty cycle and the frequency generated almost identically match the desired ones.

The duty cycle produced as a result of the control algorithms and the IGRF model has also been extensively tested. Empirical tests have been compared to the Matlab model and found to be seemingly fully functional - all tests have produced identical results to those from Matlab simulations. However at this stage we cannot say this for certain that this always is the case, as there may exist cases that produce erratic behavior such as register overflow; even though this never occurred in any of our tests. It is difficult however, to provide any illustrative static figures of these dynamic simulation. The duty cycle produced by the control algorithms in C may as well be 50%, as in the hard-coded example case. This is the main reason why all the program code is included in Appendix C.

Our next step was to measure the current running through the coil. Ideally, in order to measure a current and its relation to time, we would connect a 1Ω resistor in series with the coil and measure the voltage over it using an oscilloscope, which for 1Ω is the same as the current. Using any higher resistance

[2]The time (in per cent) that the PWM output is set to high as a fraction of the total time under consideration.

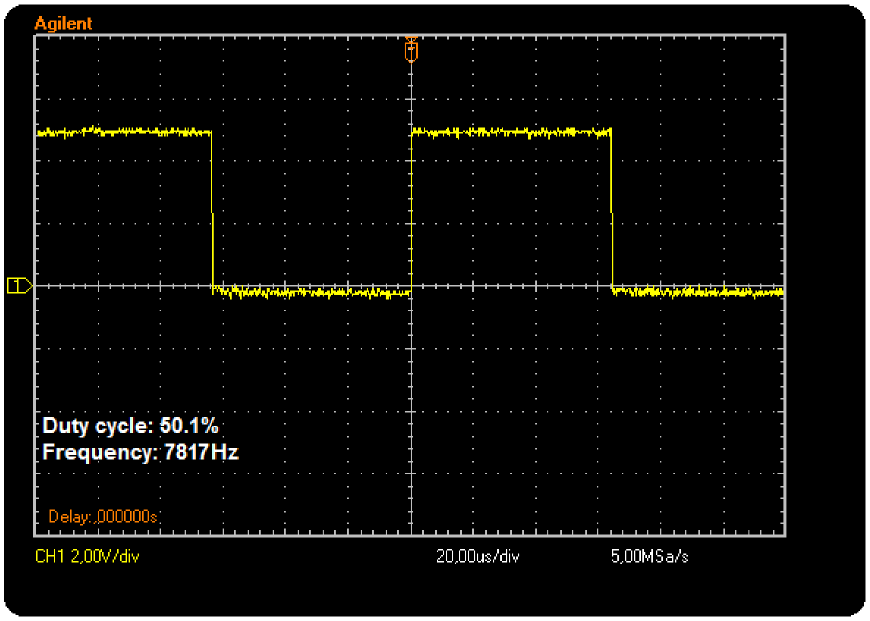


Figure 7.2: Prototype PWM test.

may alter the circuit enough to corrupt our measurements. This small resistor does unfortunately pose a serious problem when attempting to retrieve meaningful measurements. The maximum current in our coil loop being $\frac{5V}{100\Omega} = 50mA$ means we can maximally measure a voltage of $< 50mV$ over the resistor. Additionally to get any useful measurements, setting the PWM duty cycle to 50% reduces the measured voltage even further, to $< 25mV$. The oscilloscope used reaches its minimum vertical division at $20mV$, at which measurements are more or less inseparable from the additive noise; meaning that it is relatively impossible to perform any useful measurements.

To bypass the described problem, we instead insert a 10Ω resistor in series with the coil. Still using a 50% duty cycle and ignoring the coil's inductive properties we should then expect an $\frac{1}{2}\frac{5V}{(100+10)\Omega} = 22.7mA$ average current, which when measured over the resistor corresponds to $227mV$. The first measurement however, done using a 1024 bit resolution, resulted in a $11mA$ current through the coil. This indicates that the coil's inductance properties are not as negligible as previously expected, and may point to a suboptimal coil design. We observe a reduced current due to coil's time constant, which being

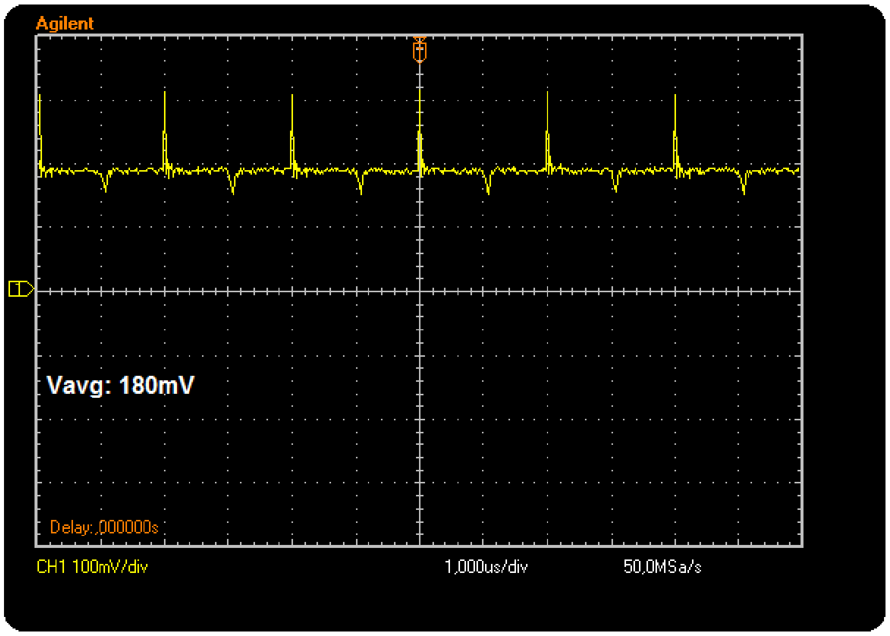


Figure 7.3: Current through the prototype coil.

at $600\mu s$ is too large relative to the PWM's period time $(\frac{16\cdot 10^6}{2\cdot 1024})^{-1}$=$128\mu s$. By reducing the coil's time constant we could attempt to match the shape of the current with that of the voltage in Figure 7.2; unfortunately this is impossible at this stage. The possible solution on the other hand is to increase the PWM's frequency. We attempt the latter by slowly reducing PWM's resolution and observing an increase in the average current, which ultimately is the only value that matters. Finally, at a resolution of 16 bits, which means that the controller can only distinguish between intervals larger than $\frac{50mA}{16} = 3.125mA$ of the desired applied current, we observe the graph in Figure 7.3. We measure a current of $\frac{180mV}{10\Omega} = 18mV$, which even at this coarse resolution is unfortunately still $4.7mA$ off the desired $22.7mA$ current. We suggest, from this strong indicator that a new coil with fewer turns should to be designed.

In order to make sure the system can run over longer time periods without overheating, a simple test was performed where we allowed the prototype to run for an hour with the PWM's duty cycle set to 100%. Under this test the prototype performed smoothly, as none of the components showed any indicators of being overloaded.

7.4 Future Work

We have attempted to create a backbone for the attitude control module design, such that those refining the module are made aware of the possible pitfalls and critical design specifications. By doing this we hope to reduce the problems this project meets in the future.

The thorn in our side throughout the whole design process has been the prototype coil and its inductance properties. Correct coil design is of utmost importance, and should be studied extensively. The coil's inductance, besides its weight, is a primary issue which relative to our designed coil should be reduced. A ferromagnetic core should also be considered once the permitted coil weight is known. According to Johannes Skaar, this may introduce a hysteresis effect that have to be accounted for.

Regarding the control algorithms, two main points remain. First is development of an algorithm which is able to switch from the detumbling controller to the reference controller at the correct time. The second second point is a deduction of the best possible gains, which can only properly be done once basic satellite parameters are established.

When it comes to the C-code implemented on the microcontroller, we find another two points where improvement is possible. A more in-depth analysis of the IGRF model is required such that we can establish what order approximation is needed. It is highly unlikely that the accuracy of a 13th order model greatly surpasses that of an e.g. 10th order model, while its runtime is 63% longer. An example of a comparison can be seen in Table 7.2. The second point is modifying the C-code to use integer variables only, in order to reduce power consumption. This may be done by multiplying all the variables and constants by e.g. 1000 thus introducing a three point accuracy, once converted to integers.

The electrical components used on the satellite need to be selected so they are suitable for operation in space; which means robustness with respect to

	x_o	y_o	z_o
Order: 13, Lat: 50, Lon: 50	13592	-4516	-15117
Order: 10, Lat: 50, Lon: 50	13600	-4513	-15133

Table 7.2: IGRF accuracy.

temperature, vacuum and radiation. Finally once the orbital parameters of the satellite are known, a function governing δ in the $\mathbf{R}_E^O$ matrix needs to be designed.

With these improvements in place, the only thing remaining, is to combine the determination and control module's C-code and implement it on one controller. This should be relatively straight-forward. The reason why we did not do this ourselves is because it would not improve the understanding for those continuing the project, and we ran out of time to perform any relevant tests. We did however solder the sensor connectors, which are not included in Figure A.9.

We wish good luck to those continuing this project.

Appendix A

Additional Figures

This appendix provides additional figures from simulations performed in Chapter 5 together with some explanations and the prototype connection setup illustrations.

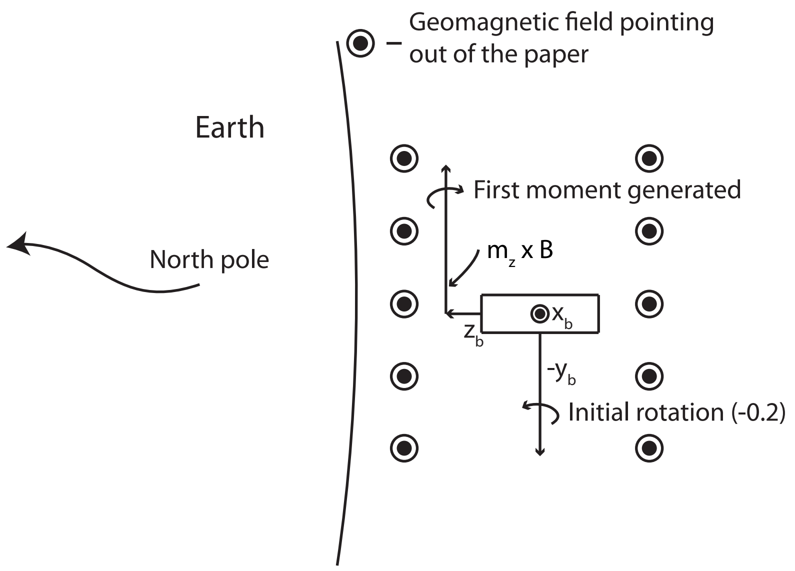


Figure A.1: Coil generated moment illustration. Earth viewed from above the north pole.

A.1 Detumbling Simulation

We start by providing the reasoning for the moments generated by the coils in Figure 5.3. Moments from the coils can be rather tedious to keep track of in three dimensions, so in order to explain them we were forced to select a simple initial condition. Figure A.1 illustrates the scenario from Section 5.2. The satellite is perfectly aligned with the orbit frame and has a $0.2\frac{\text{rad}}{s}$ rotation about the negative y axis. The geomagnetic field illustrated by the circles is coming out of the page. The controller dissipates the system's energy by applying a current through the z^+ face, generating a moment out of the z^+ face coil. This moment m_z crossed by the geomagnetic B field creates a torque in the opposite direction of the satellite's rotation, thus dissipating its kinetic energy. The x_b and z_b axes are however not constant relative to the orbit frame, while $\boldsymbol{\omega}_{OB}^{B} \neq \mathbf{0}$. This is why in Figure 5.3 the moments from the z^+ and x^+ coils are identical apart from the phase shift. After the satellite makes a $\frac{\pi}{2}$ rotation about the $-y$ axis, the x_b axis is where the z_b axis originally was. At this point the controller applies the same moment out of the x^+ face coil as it initially did out of the z^+ coil.
Beneath follow the rest of the figures from the detumbling controller simulations.

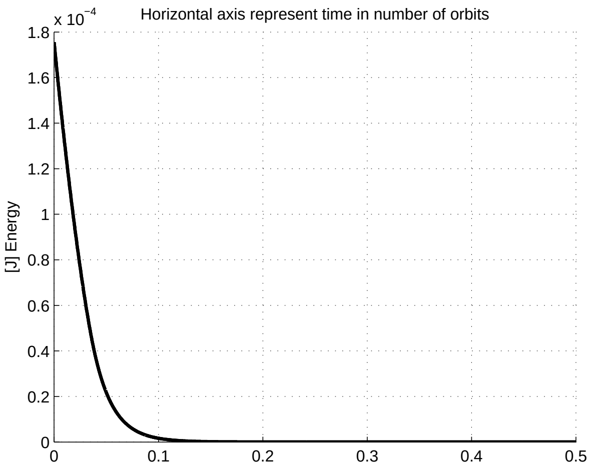


Figure A.2: Dissipation of satellite's kinetic energy.

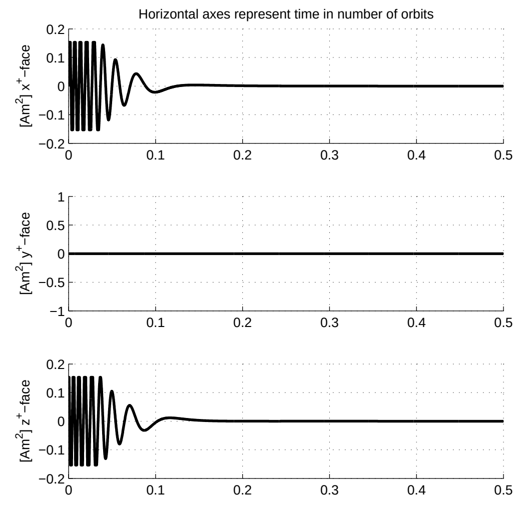


Figure A.3: Full figure of the moments generated by the coils in the dissipative detumbling controller simulation.

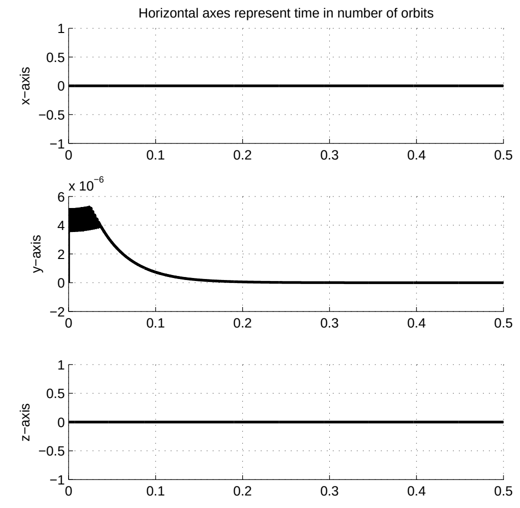


Figure A.4: Full figure of the torques generated by the coils in the dissipative detumbling controller simulation.

A.2 Reference Simulation

This section includes the rest of the figures from the reference controller simulation.

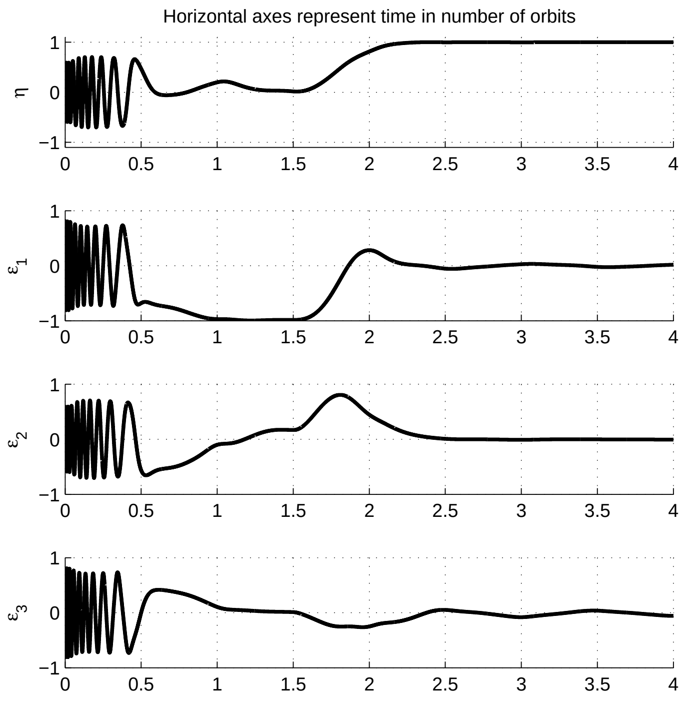


Figure A.5: Quaternions from the full detumbling and reference control simulation.

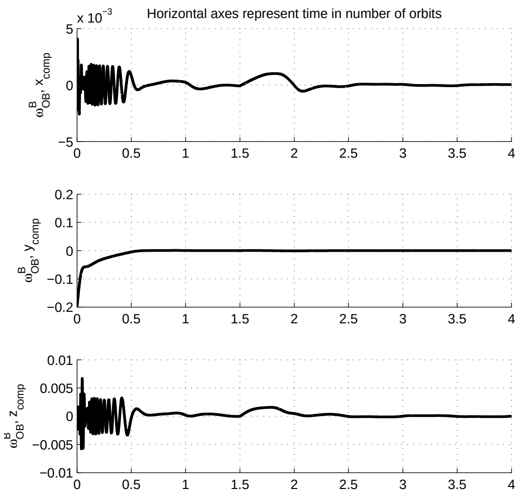


Figure A.6: $\boldsymbol{\omega}^B_{OB}$ from the full detumbling and reference control simulation.

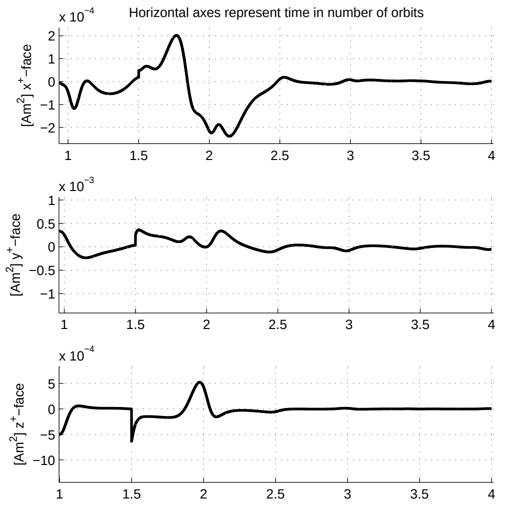


Figure A.7: Moments from the coils, enhanced to show dynamics from the reference controller starting after 1.5 orbits.

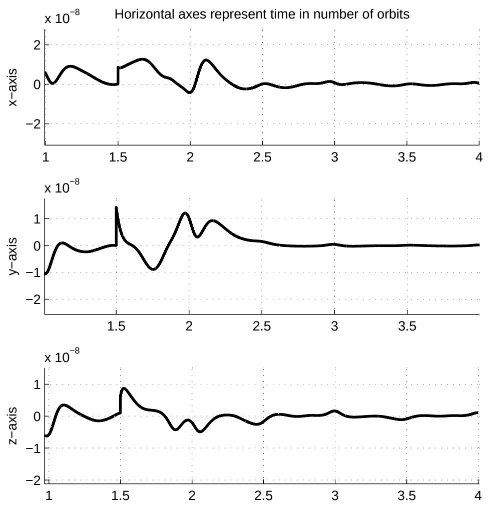


Figure A.8: Torques on the satellite, enhanced to show dynamics from the reference controller starting after 1.5 orbits.

A.3 Prototype Connection

Connection setup for programming and running the prototype can be seen in A.9. Note the orientation and placement of the JTAG and power supply cables, especially the missing pin on the power supply which is connectionless. Figure A.10 on the other hand, shows a full connection-measurement setup, when simulating the system in its entirety. A resistor, in parallel with the probe, should be added in series with the coil when measuring the current.

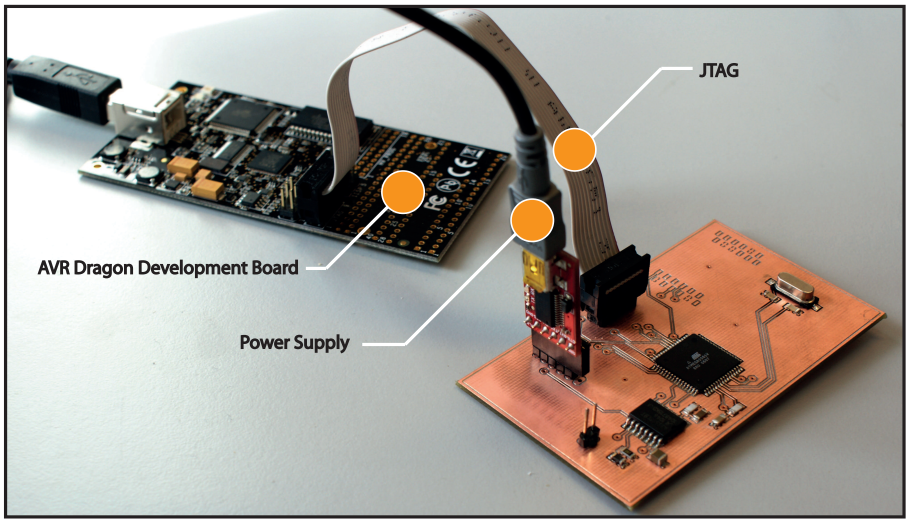


Figure A.9: Prototype connection.

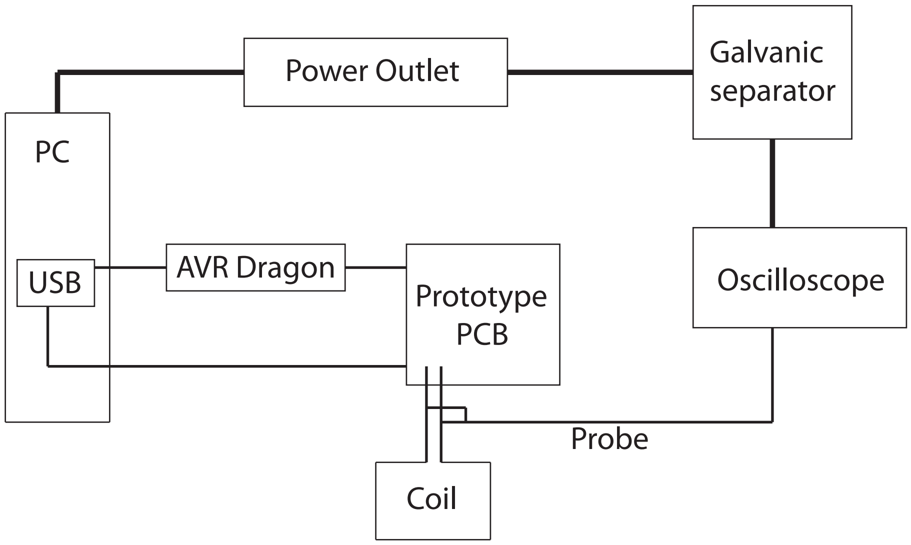
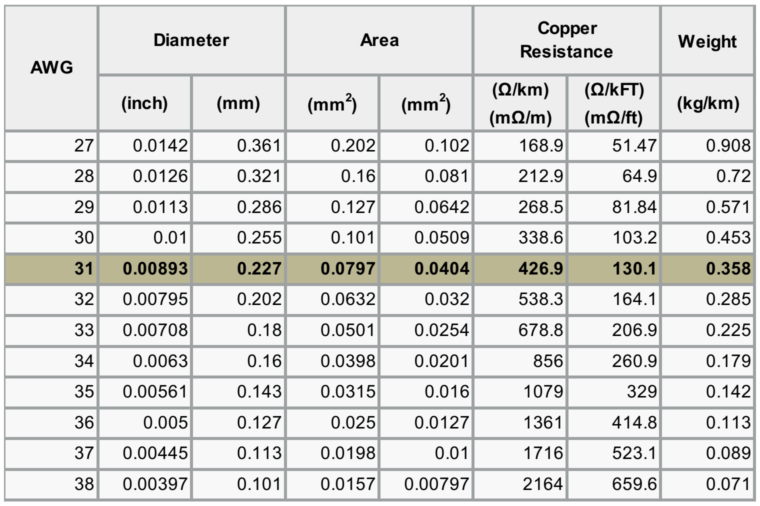


Figure A.10: Full prototype connection for programming and taking measurements.

Appendix B

Standards, Tables and IGRF

Figure B.1 shows the American Wire Gauge chart used under coil design.



AWG	Diameter		Area		Copper Resistance		Weight
	(inch)	(mm)	(mm^2)	(mm^2)	(Ω/km) (mΩ/m)	(Ω/kFT) (mΩ/ft)	(kg/km)
27	0.0142	0.361	0.202	0.102	168.9	51.47	0.908
28	0.0126	0.321	0.16	0.081	212.9	64.9	0.72
29	0.0113	0.286	0.127	0.0642	268.5	81.84	0.571
30	0.01	0.255	0.101	0.0509	338.6	103.2	0.453
31	**0.00893**	**0.227**	**0.0797**	**0.0404**	**426.9**	**130.1**	**0.358**
32	0.00795	0.202	0.0632	0.032	538.3	164.1	0.285
33	0.00708	0.18	0.0501	0.0254	678.8	206.9	0.225
34	0.0063	0.16	0.0398	0.0201	856	260.9	0.179
35	0.00561	0.143	0.0315	0.016	1079	329	0.142
36	0.005	0.127	0.025	0.0127	1361	414.8	0.113
37	0.00445	0.113	0.0198	0.01	1716	523.1	0.089
38	0.00397	0.101	0.0157	0.00797	2164	659.6	0.071

Figure B.1: American Wire Gauge standard.

The IGRF-11 coefficients used in this thesis are extracted from [16] and presented in Table B.1. The online available coefficients can be downloaded as a text or an Excel file. We recommend using an Excel file as we have created a simple Matlab script that extracts these coefficients and creates a header file that can be used in the C-program to define the coefficients. This script can be seen in Appendix C as $IGRF11\_load\_data.m$ which takes an Excel file having a single pillar structure as in Table B.2; extracts the

IGRF coefficients and stores them as Matlab variables such that they can be used for further simulations. Further on, it reads these Matlab variables and stores their data into a *IGRFcoeffs.h* file, ready to be used as a C-header containing the IGRF coefficients. We created this to avoid erranous coefficients and simplify future header coefficient updates.

g/h	n	m	IGRF	SV	g/h	n	m	IGRF	SV	g/h	n	m	IGRF	g/h	n	m	IGRF	g/h	n	m	IGRF
g	1	0	-29496.5	11.4	g	6	3	-141.4	1.9	g	9	0	5.4	g	11	0	3.0	g	12	9	-0.4
g	1	1	-1585.9	16.7	h	6	3	61.5	-0.4	g	9	1	9.4	g	11	1	-1.5	h	12	9	0.3
h	1	1	4945.1	-28.8	g	6	4	-22.9	-1.6	h	9	1	-20.5	h	11	1	0.1	g	12	10	0.2
g	2	0	-2396.6	-11.3	h	6	4	-66.3	-0.5	g	9	2	3.4	g	11	2	-2.1	h	12	10	-0.9
g	2	1	3026.0	-3.9	g	6	5	13.1	-0.2	h	9	2	11.6	h	11	2	1.7	g	12	11	-0.8
h	2	1	-2707.7	-23.0	h	6	5	3.1	0.8	g	9	3	-5.3	g	11	3	1.6	h	12	11	-0.2
g	2	2	1668.6	2.7	g	6	6	-77.9	1.8	h	9	3	12.8	h	11	3	-0.6	g	12	12	0.0
h	2	2	-575.4	-12.9	h	6	6	54.9	0.5	g	9	4	3.1	g	11	4	-0.5	h	12	12	0.8
g	3	0	1339.7	1.3	g	7	0	80.4	0.2	h	9	4	-7.2	h	11	4	-1.8	g	13	0	-0.2
g	3	1	-2326.3	-3.9	g	7	1	-75.0	-0.1	g	9	5	-12.4	g	11	5	0.5	g	13	1	-0.9
h	3	1	-160.5	8.6	h	7	1	-57.8	0.6	h	9	5	-7.4	h	11	5	0.9	h	13	1	-0.8
g	3	2	1231.7	-2.9	g	7	2	-4.7	-0.6	g	9	6	-0.8	g	11	6	-0.8	g	13	2	0.3
h	3	2	251.7	-2.9	h	7	2	-21.2	0.3	h	9	6	8.0	h	11	6	-0.4	h	13	2	0.3
g	3	3	634.2	-8.1	g	7	3	45.3	1.4	g	9	7	8.4	g	11	7	0.4	g	13	3	0.4
h	3	3	-536.8	-2.1	h	7	3	6.6	-0.2	h	9	7	2.2	h	11	7	-2.5	h	13	3	1.7
g	4	0	912.6	-1.4	g	7	4	14.0	0.3	g	9	8	-8.4	g	11	8	1.8	g	13	4	-0.4
g	4	1	809.0	2.0	h	7	4	24.9	-0.1	h	9	8	-6.1	h	11	8	-1.3	h	13	4	-0.6
h	4	1	286.4	0.4	g	7	5	10.4	0.1	g	9	9	-10.1	g	11	9	0.2	g	13	5	1.1
g	4	2	166.6	-8.9	h	7	5	7.0	-0.8	h	9	9	7.0	h	11	9	-2.1	h	13	5	-1.2
h	4	2	-211.2	3.2	g	7	6	1.6	-0.8	g	10	0	-2.0	g	11	10	0.8	g	13	6	-0.3
g	4	3	-357.1	4.4	h	7	6	-27.7	-0.3	g	10	1	-6.3	h	11	10	-1.9	h	13	6	-0.1
h	4	3	164.4	3.6	g	7	7	4.9	0.4	h	10	1	2.8	g	11	11	3.8	g	13	7	0.8
g	4	4	89.7	-2.3	h	7	7	-3.4	0.2	g	10	2	0.9	h	11	11	-1.8	h	13	7	0.5
h	4	4	-309.2	-0.8	g	8	0	24.3	-0.1	h	10	2	-0.1	g	12	0	-2.1	g	13	8	-0.2
g	5	0	-231.1	-0.5	g	8	1	8.2	0.1	g	10	3	-1.1	g	12	1	-0.2	h	13	8	0.1
g	5	1	357.2	0.5	h	8	1	10.9	0.0	h	10	3	4.7	h	12	1	-0.8	g	13	9	0.4
h	5	1	44.7	0.5	g	8	2	-14.5	-0.5	g	10	4	-0.2	g	12	2	0.3	h	13	9	0.5
g	5	2	200.3	-1.5	h	8	2	-20.0	0.2	h	10	4	4.4	h	12	2	0.3	g	13	10	0.0
h	5	2	188.9	1.5	g	8	3	-5.7	0.3	g	10	5	2.5	g	12	3	1.0	h	13	10	0.4
g	5	3	-141.2	-0.7	h	8	3	11.9	0.5	h	10	5	-7.2	h	12	3	2.2	g	13	11	0.4
h	5	3	-118.1	0.9	g	8	4	-19.3	-0.3	g	10	6	-0.3	g	12	4	-0.7	h	13	11	-0.2
g	5	4	-163.1	1.3	h	8	4	-17.4	0.4	h	10	6	-1.0	h	12	4	-2.5	g	13	12	-0.3
h	5	4	0.1	3.7	g	8	5	11.6	0.3	g	10	7	2.2	g	12	5	0.9	h	13	12	-0.5
g	5	5	-7.7	1.4	h	8	5	16.7	0.1	h	10	7	-4.0	h	12	5	0.5	g	13	13	-0.3
h	5	5	100.9	-0.6	g	8	6	10.9	0.2	g	10	8	3.1	g	12	6	-0.1	h	13	13	-0.8
g	6	0	72.8	-0.3	h	8	6	7.1	-0.1	h	10	8	-2.0	h	12	6	0.6				
g	6	1	68.6	-0.3	g	8	7	-14.1	-0.5	g	10	9	-1.0	g	12	7	0.5				
h	6	1	-20.8	-0.1	h	8	7	-10.8	0.4	h	10	9	-2.0	h	12	7	0.0				
g	6	2	76.0	-0.3	g	8	8	-3.7	0.2	g	10	10	-2.8	g	12	8	-0.4				
h	6	2	44.2	-2.1	h	8	8	1.7	0.4	h	10	10	-8.3	h	12	8	0.1				

Table B.1: 11th Generation IGRF Schmidt semi-normalised spherical harmonic coefficients in nT. SV for all last three column sets is zero.

g	1	0	-29496.5	11.4
g	1	1	-1585.9	16.7
$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$	$\downarrow$

Table B.2: Coefficient Excel file structure for C-header generation.

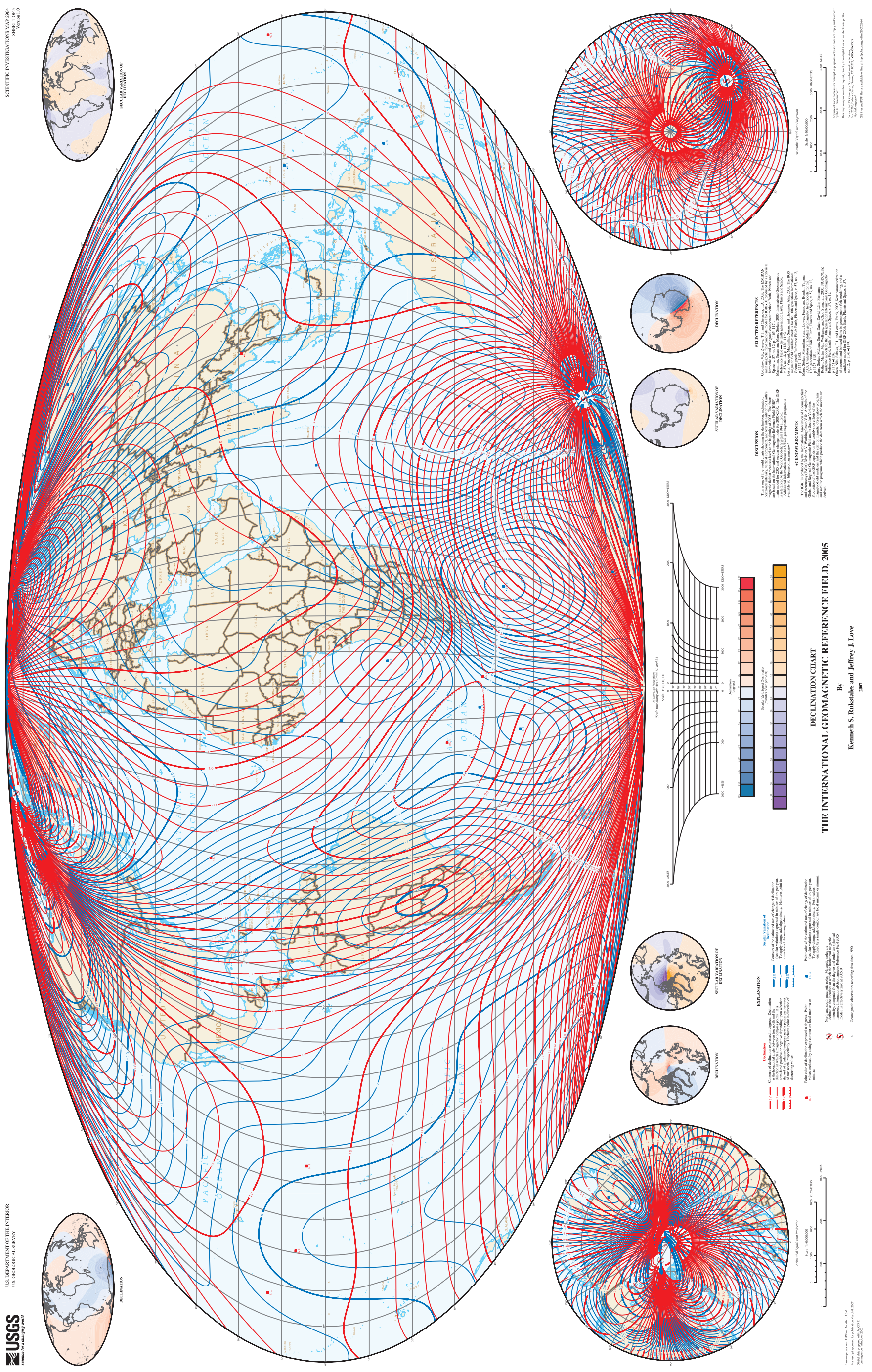


Figure B.2: Illustration of the 2005 IGRF model. Source: [19].

Appendix C

Source Code

Code file description	
main.m	Main script, executed manually.
parameters.m	Function used to generate struct P containing all the necessary simulation parameters. Executed by main.m.
nonLinearSatellite.m	The system with all its differential equations. Executed through ODE45 by main.m.
IGRF.m	Solves the IGRF equations to generate the magnetic field vector $\mathbf{B}^O$. Executed by nonlinearSatellite.m.
eul2qua.m	Converts $\boldsymbol{\omega}_{OB}^B$ and Euler parameters to $\boldsymbol{\omega}_{IB}^B$ and quaternions. Executed by main.m to find initial conditions.
qua2eul.m	Inverse of eul2qua.m. To be excuted manually for plotting purposes.
outFcn.m	ODE45 out function, used to store variables on successfull ODE45 runs. Executed implicitly
IGRF11_load_data.m	Creates a matlab variable with IGRF coefficients. Can optionally create a C header file. Executed manually
attitudeControl.c	Program with the main loop for attitude control. To be implemented on the microcontroller.
IGRFcoeffs.h	Header file with IGRF coefficients and a few other constant parameters needed by attitudeControl.c

Table C.1: Source code summary.

C.1 Matlab Code

```
%*******************************************************************
% File main.m
% Main script. Simulates (integrates) the systems.
%
% Written by Zdenko Tudor, 2011
%*******************************************************************

clear all; clc;
addpath('myIGRF');

% Initialize storage variables (For testing and plotting purposes only)
global VAR;

% Create parameters struct
P = parameters();

% Load neccessary parameters
orbitPeriod = P.orbitPeriod;    % For defining simulation length
scalePlot = P.scalePlot;        % For x-axes scaling when plotting

% Initial and simulation parameters
numOfOrbits = 4;                % Simulation time in number of orbits
tSpan = [0,orbitPeriod*numOfOrbits];% Time span for ODE45

w_B_OB = [0;-0.2;0];                        % Initial rotational velocity
eulAng = [80;50;170];                       % Initial orientation
[w_B_IB,qua] = eul2qua(P,w_B_OB',eulAng');% Transform initial parameters
initLat = 0;                                % Initial latitude
initJoule = 0;                              % Always to be set to zero

% State vector x = [wx wy wz eta eps1 eps2 eps3 latitude totalJoule];
xInit = [w_B_IB;qua;initLat;initJoule];

% ODE45 simulation settings for faster results
% options = odeset('MaxStep',5,'OutputFcn',@outFcn,'Refine',1);
% ODE45 simulation settings for precice results
options = odeset('MaxStep',0.2,'OutputFcn',@outFcn,'RelTol',1e-12,'
  AbsTol',1e-12,'Refine',1);

% Simulate system
[tout,yout] = ode45(@(t,x)nonlinearSatellite(t,x,P),tSpan,xInit,options)
  ;
```

```
%*********************************************************************
% File parameters.m
%
% P = parameters()
%
% Creates a parameter struct P, which is used to pass the parameters
% between different functions.
%
% Written by Zdenko Tudor, 2011
%*********************************************************************

function P = parameters()
% Coil data
% Battery voltage
V = 5;

% Number of coil windings (turns)
Nx = 355;
Ny = 355;
Nz = 800;

% [m^2] Coil area
Ax = 0.08*0.18;
Ay = 0.08*0.18;
Az = 0.08^2;

% [Ohm] Coil resistance
Rx = 110;
Ry = 110;
Rz = 110;

% [A] Maximum current through coils
ix_max = V/Rx;
iy_max = V/Ry;
iz_max = V/Rz;

% Define satelite animation object
satX=[0 1 1 0;0 1 1 0;0 0 0 0;0 1 1 0; 0 1 1 0;1 1 1 1];
satY=[0 0 1 1;0 0 0 0;0 1 1 0;0 0 1 1; 1 1 1 1;0 1 1 0];
satZ=[0 0 0 0;0 0 1 1;0 0 1 1;1 1 1 1; 0 0 1 1;0 0 1 1]*2;

% Move mass center to reference frame center
satX = satX-ones(6,4)./2;
satY = satY-ones(6,4)./2;
satZ = satZ-ones(6,4);
```

```

m = 2;                          % [kg] Satellite mass
dx = 0.11;                      % X-axis length
dy = 0.1;                       % Y-axis length
dz = 0.2;                       % Z-axis length
Re = 6371.2e3;                  % [m] Earth radius
Rs = 600e3;                     % [m] Satellite altitude
Rc = Re+Rs;                     % [m] Distance from earth center to
    satellite
Ix = (m/12)*(dy^2+dz^2);        % X-axis inertia
Iy = (m/12)*(dx^2+dz^2);        % Y-axis inertia
Iz = (m/12)*(dx^2+dy^2);        % Z-axis inertia

I = diag([Ix Iy Iz]);           % Inertia matrix

G = 6.67428e-11;                % Earth gravitational constant
M = 5.972e24;                   % Earth mass

w_o = sqrt(G*M/Rc^3);           % Satellite angular velocity relative Earth

orbitPeriod = (2*pi)/(w_o);         % For defining simulation length
scalePlot = 1/orbitPeriod;          % For x-axes scaling when plotting

% Store variables
P.satX = satX; P.satY = satY; P.satZ = satZ;
P.w_o = w_o;
P.V = V; P.I = I;
P.Nx = Nx; P.Ny = Ny; P.Nz = Nz;
P.Ax = Ax; P.Ay = Ay; P.Az = Az;
P.Rx = Rx; P.Ry = Ry; P.Rz = Rz;
P.ix_max = ix_max; P.iy_max = iy_max; P.iz_max = iz_max;
P.Re = Re; P.Rc = Rc;
P.orbitPeriod = orbitPeriod; P.scalePlot = scalePlot;
end
```

```
%*****************************************************************
% File nonlinearSatellite.m
%
% xDot = nonlinearSatellite(T,X,P)
%
% Simulates the earth-satellite system. To be used with an ordinary
% differential equation solver such as MATLAB's Runge Kutta(4,5) method,
% ODE45.
% The input:
% T - time vector (implicit)
% X - state vector, initial state has to be provided. The state vector
  is
% of the form X = [W^B_IB' Q' LAMBDA JOULE]. W^B_IB is a 3x1 vector
% containing the three rotational velocity components. Q is the
  quaternion
% vector of the form Q = [eta eps1 eps2 eps3]. LAMBDA is the latitude of
% the satellite and JOULE is only a testing paramemter which will return
% the total Joule consumption of the satellite during the simulation.
% Initial JOULE should always be set to zero.
% P - struct containg necessary parameters (w_o, I, Rc, coil data,
  voltage
% and current ratings).
%
% Example:
% [TOUT,YOUT] = ode45(@(t,x)nonlinearSatellite(t,x,P),tSpan,xInit,
  options)
% this produces:
% TOUT - time vector for the integration
% YOUT - Integrated xDot vector for each timestep size(YOUT)=length(TOUT
  )x9
%
% Written by Zdenko Tudor, 2011
%*****************************************************************

function xDot = nonlinearSatellite(t,x,P)
global R_B_O;

% Load necessary parameters from P struct.
w_o = P.w_o;
I = P.I;

% Normalization of quartenions
x(4:7) = x(4:7)./norm(x(4:7));

% State vector x = [];
```

```
w_B_IB = x(1:3);                % Angular velocity vector
eta = x(4);                     % Euler parameter eta
eps = x(5:7);                   % Euler parameter epsilon
lat = x(8)                      % Lambda

% Epsilon crossmatrix, skew-symmetric
S_eps = [ 0, -eps(3), eps(2);
          eps(3), 0, -eps(1);
         -eps(2), eps(1), 0];

% Rotation matrices
R_O_B = eye(3) + 2*eta*S_eps + 2*S_eps^2;
R_B_O = R_O_B';

% Angular velocities of frames relative each other
w_O_IO = [0;-w_o;0];
w_B_OB = w_B_IB-R_B_O*w_O_IO;

% Switch controller after this many orbits
orbitSwitch = 1;

% Torque from controllers. Logic 'if' is for controller switching.
if t<P.orbitPeriod*orbitSwitch
    % Pick detumbling controller
    [tau_m,j] = detumblingController1(P,w_B_OB,lat);
%    [tau_m,j] = detumblingController2(P,w_B_OB,lat);
else
    [tau_m,j] = referenceController(P,w_B_OB,lat,eps);
end

%-----Storing variables used for testing purposes only-----Start-----
global tmpVAR;
tmpVAR.torque = tau_m;
xDot(9,1) = j;
%-----Storing variables used for testing purposes only------End------

% Gravitational torque
c3 = R_B_O(:,3);
tau_g = 3*w_o^2*cross(c3,I*c3);

% Angular acceleration
wDot_B_IB=I\(tau_m+tau_g-cross(w_B_IB,I*w_B_IB));

% Eta
etaDot = -0.5*eps'*w_B_OB;
```

```

% Epsilon
epsDot = 0.5*(eta*eye(3)+S_eps)*w_B_OB;

% Variable assignment
xDot(1:3,1) = wDot_B_IB;
xDot(4,1) = etaDot;
xDot(5:7,1) = epsDot;
xDot(8,1) = w_o;
end

% Dissipative non-linear detumbling controller.
% Dissipates the kinetic energy from the satellite until w^B_OB == [0 0
    0].
function [tau_m,j] = detumblingController1(P,w_B_OB,lat)
global R_B_O;
% Geomagnetic field in Orbit frame and in Body frame, pick diple or IGRF
    .
% [B_O,~] = dipoleField(P,lat);
[B_O] = IGRF(10,10,lat,0,P);
B_B = R_B_O*B_O;

% Controller gain
d = 4e-5;

% Moment set up by coils before scaling (saturating current)
m_B = -(d/norm(B_B,2)^2)*cross(B_B,w_B_OB);

% Moment set up by coils after scaling
[m_B,j] = currentScaling(P,m_B);

% Torque set up by coils
tau_m = cross(m_B,B_B);
end

% Bdot detumbling controller
% Controls the satellite to follow the geomagnetic field
function [tau_m,j] = detumblingController2(P,w_B_OB,lat)
global R_B_O;
% Geomagnetic field in Orbit frame and in Body frame
[B_O,Bdot_O] = dipoleField(P,lat);
B_B = R_B_O*B_O;
S_w = [0 -w_B_OB(3) w_B_OB(2);
       w_B_OB(3) 0 -w_B_OB(1);
       -w_B_OB(2) w_B_OB(1) 0];
```

```
Bdot_B = -S_w*R_B_O*B_O+R_B_O*Bdot_O;

% Controller gain
k = 4e-5;

% Moment set up by coils before scaling (saturating current)
m_B = (-k/norm(B_B,2)^2)*Bdot_B;

% Moment set up by coils after scaling
[m_B,j] = currentScaling(P,m_B);

% Torque set up by coils
tau_m = cross(m_B,B_B);
end

% Satellite reference controller for large deviations from R_B_O = eye
  (3).
function [tau_m,j] = referenceController(P,w_B_OB,lat,eps)
global R_B_O;
% Geomagnetic field in Orbit frame and in Body frame
% [B_O,~] = dipoleField(P,lat);
[B_O] = IGRF(10,10,lat,0,P);
B_B = R_B_O*B_O;

% Controller gains
d = 4e-5;
k = 5e-8;

% Moment set up by coils before scaling (saturating current)
m_B = (1/norm(B_B,2)^2)*(-d*cross(B_B,w_B_OB)-k*cross(B_B,eps));

% Moment set up by coils after scaling
[m_B,j] = currentScaling(P,m_B);

% Torque set up by coils
tau_m = cross(m_B,B_B);
end

% Scales (saturates) the power consumption if the maximum currents are
% exceeded
function [m_B,j] = currentScaling(P,m_B)
% Load coil parameters from parameter struct
Nx = P.Nx; Ny = P.Ny; Nz = P.Nz;
Ax = P.Ax; Ay = P.Ay; Az = P.Az;
ix_max = P.ix_max; iy_max = P.iy_max; iz_max = P.iz_max;
```

```
V = P.V;


% Current scaling
% Creates ratio variable containing a coil's: max current/wanted current
    .
% The lowest value in ratio variable, if below 1 is the highest current
% violation, thus all the currents should be scaled by this factor.
ix = m_B(1)/(Nx*Ax);  ratio(1)=ix_max/abs(ix);
iy = m_B(2)/(Ny*Ay);  ratio(2)=iy_max/abs(iy);
iz = m_B(3)/(Nz*Az);  ratio(3)=iz_max/abs(iz);

if min(ratio)<1
    m_B = m_B*min(ratio);
end

% Power consumption (Joules)
Wx = abs(V*m_B(1)/(Nx*Ax));
Wy = abs(V*m_B(2)/(Ny*Ay));
Wz = abs(V*m_B(3)/(Nz*Az));
j = Wx+Wy+Wz;

%-----Storing variables used for testing purposes only-----Start-----
global tmpVAR;
tmpVAR.moment=m_B;
tmpVAR.W = Wx+Wy+Wz;
%-----Storing variables used for testing purposes only------End------
end

% Returns the dipole model of the geomagnetic field in Teslas,
    represented
% in the Orbit frame.s
function [B_O,Bdot_O] = dipoleField(P,lat)
w_o = P.w_o;
Rc = P.Rc;

my_m = 1e17/(4*pi);
B0 = my_m/Rc^3;

Bn = B0*cos(lat);        % x-axis in orbital frame, parallel to the
    velocity vector
Br = 2*B0*sin(lat);      % z-axis in orbital frame, toward Earth center
B_O = [Bn; 0; Br];

BnDot = -w_o*B0*sin(lat);
```

```
BrDot = w_o*2*B0*cos(lat);
Bdot_O = [BnDot; 0; BrDot];
end
```

```
%******************************************************************
% File IGRF.m
%
% Algorithm for finding the magnetic field vectors from the IGRF-11
% model for a given latitude LAT and longitude LON in radians, with
% order n and degree m. The output is the magnetic field vector
% represented in the orbit frame.
%
% Created by Raymond Kristiansen, 2000
% Edited by Zdenko Tudor, 2011
%******************************************************************
function B = IGRF(n, m, lat, lon, P)

% Load IGRF11 coefficient variables
load('IGRF11_data')

i = n;
j = m;

% Fix latitude in case of possible singularity
mt = mod(lat,2*pi);
tol = 1e-9;
if (mt>(pi/2)-tol && mt<(pi/2)+tol)
     lat=(pi/2)+tol;
elseif(mt>(3*pi/2)-tol && mt<(3*pi/2)+tol)
     lat=(3*pi/2)+tol;
end

% Defining constants
Re = P.Re;
Rc = P.Rc;

% Calculating colatitude and longitude in radians
theta = (pi/2)-lat;
phi = lon;

% Zero offset
O = 1;

% Defining temporary variables
Bt2 = 0; Bp2 = 0; Br2 = 0;

% Calculating Legendre polynominals
[P, dP, S]=Pfunk(n, m, theta);

```

```
% Calculating field vectors
for n=1:i
    Bt1 = 0;
    Bp1 = 0;
    Br1 = 0;
    for m=0:j
        Bt1 = Bt1 + (S(O+n,O+m)*g_data(O+n,O+m)*cos(m*phi)+S(O+n,O+m)
          *...
            h_data(O+n, O+m)*sin(m*phi))*dP(O+n,O+m);
        Bp1 = Bp1 + (m*S(O+n,O+m)*h_data(O+n, O+m)*cos(m*phi)-m*...
            S(O+n,O+m)*g_data(O+n, O+m)*sin(m*phi))*P(O+n,O+m);
        Br1 = Br1 + (S(O+n,O+m)*g_data(O+n,O+m)*cos(m*phi)+S(O+n,O+m)
          *...
            h_data(O+n, O+m)*sin(m*phi))*P(O+n,O+m);
    end
    Bt2 = Bt2 + ((Re/Rc)^(n+2))*Bt1;
    Bp2 = Bp2 + ((Re/Rc)^(n+2))*Bp1;
    Br2 = Br2 + (n+1)*((Re/Rc)^(n+2))*Br1;
end

% In Cartesian coordinates
eps = 0;
X = Bt2*cos(eps)-Br2*sin(eps);
Y = -Bp2/(sin(theta));
Z = -Bt2*sin(eps)-Br2*cos(eps);

B = [X;Y;Z]*1e-9;
end

%
  ************************************************************************

%
% Algorithm for calculating the associated Legendre polynominals
% for the given order n, degree m and co-latitude theta. The output
% is the Legendre polynominal P, its partial derivative dP and
% the Schmidt normalization matrix S.
%
% Created by Raymond Kristiansen, 2000
% Edited by Zdenko Tudor, 2011
%
  ************************************************************************

function [P, dP, S] = Pfunk(n, m, theta)
O = 1;  % Zero index offset
```

```

% Defining zero matrices
P = ones(n+O, m+O);
dP = zeros(n+O, m+O);
S = ones(n+O, m+O);
i = n;
j = m;

% Calculating S matrix
for n = 0:1:i
    for m = 0:1:j
        if n > 0
            if m == 0
                S(O+n, O+m) = S(n-1+O, 0+O)*(2*n-1)/n;
            else
                if m == 1
                    deltaFun = 1;
                else
                    deltaFun = 0;
                end
                S(n+O, m+O) = S(n+O, m-1+O)*sqrt(((n-m+1)*(deltaFun+1))/(n+m));
            end
        end
    end
end

% Calculates the Legendre polynominal P and its partial derivative dP
for n = 0:1:i
    for m = 0:1:j
        if n==1
            dP(n+O, m+O) = cos(theta)*dP(n-1+O, m+O)-sin(theta)*P(n-1+O, m+O);
            P(n+O, m+O) = cos(theta)*P(n-1+O, m+O);
        elseif n>1
            K=((n-1)^2-m^2)/((2*n-1)*(2*n-3));
            dP(n+O, m+O) = cos(theta)*dP(n-1+O, m+O)-sin(theta)*P(n-1+O, m+O) -...
                K*dP(n-2+O, m+O);
            P(n+O, m+O) = cos(theta)*P(n-1+O, m+O) - K*P(n-2+O, m+O);
        end
        if n==m && n>0
            P(n+O, m+O) = sin(theta)*P(n-1+O, n-1+O);
            dP(n+O, m+O) = sin(theta)*dP(n-1+O, n-1+O)+cos(theta)*P(n-1+O, n-1+O);
            end
      end
end
end
```

```
%*******************************************************************
% File eul2qua.m
%
% [W_B_IB,QUA] = eul2qua(P,W_B_OB,EUL)
%
% Function used for converting Euler angles to quaternions. This
  function
% also calculates the angular velocity between B and I represented in B.
%
% Input:
% P - struct containg necessary parameter(s) (w_o)
% W_B_OB - Is W^B_OB, a vector containing the three rotational velocity
% components. Size 3x1
% EUL - Vector containing Euler angles. Size 3x1. Form: [phi, theta, psi
  ]'.
% Angles are in degrees.
%
% Output:
% W_B_IB - Is W^B_IB, a vector containing the three rotational velocity
% components. Size 3x1
% QUA - Vector containing quaternions. Size 4x1.
% Form: [eta eps1 eps2 eps3]'.
%
% Example:
% [w_B_OB, eul] = qua2eul(P,yout(:,1:3),yout(:,4:7));
%
% Written by Per Kolbjørn Soglo, 1994
% Slightly edited by Zdenko Tudor, 2011
%*******************************************************************
function [w_B_IB,qua] = eul2qua(P,w_B_OB,eul)
% Load neccessary parameter
w_o=P.w_o;

% Conversion from degrees to radians
eul(1) = pi*eul(1)/180;
eul(2) = pi*eul(2)/180;
eul(3) = pi*eul(3)/180;

% Rotation matrix from O to B
c1 = cos(eul(1)); c2 = cos(eul(2)); c3 = cos(eul(3));
s1 = sin(eul(1)); s2 = sin(eul(2)); s3 = sin(eul(3));

% XYZ-rotation
R11 = c2*c3;
R12 = -c2*s3;
```

```
R13 = s2;
R21 = c1*s3+c3*s1*s2;
R22 = c1*c3-s1*s2*s3;
R23 = -c2*s1;
R31 = s1*s3-c1*c3*s2;
R32 = c1*s2*s3+c3*s1;
R33 = c1*c2;
R_O_B = [R11 R12 R13
         R21 R22 R23
         R31 R32 R33];

% Calculating angular velocity between O and I represented in B
w_B_IO(1) = -w_o*R21;
w_B_IO(2) = -w_o*R22;
w_B_IO(3) = -w_o*R23;

% Calculating angular velocity between B and I represented in B
w_B_IB(1) = w_B_OB(1) + w_B_IO(1);
w_B_IB(2) = w_B_OB(2) + w_B_IO(2);
w_B_IB(3) = w_B_OB(3) + w_B_IO(3);
w_B_IB = w_B_IB';

% Calculating Euler parameters
R44 = trace(R_O_B);
maxVal = max([R11 R22 R33 R44]);
if R11 == maxVal
    p1 = sqrt(1 + 2*R11-R44);
    p2 = (R21 + R12)/p1;
    p3 = (R13 + R31)/p1;
    p4 = (R32 - R23)/p1;
elseif R22 == maxVal
    p2 = sqrt(1 + 2*R22-R44);
    p1 = (R21 + R12)/p2;
    p3 = (R32 + R23)/p2;
    p4 = (R13 - R31)/p2;
elseif R33 == maxVal
    p3 = sqrt(1 + 2*R33-R44);
    p1 = (R13 + R31)/p3;
    p2 = (R32 + R23)/p3;
    p4 = (R21 - R12)/p3;
else
    p4 = sqrt(1 + R44);
    p1 = (R32 - R23)/p4;
    p2 = (R13 - R31)/p4;
    p3 = (R21 - R12)/p4;
```

```
end

qua(1) = p4/2;
qua(2) = p1/2;
qua(3) = p2/2;
qua(4) = p3/2;
qua = qua';
end
```

```
%*****************************************************************
% File qua2eul.m
%
% [W_B_OB,EUL] = qua2eul(P,W_B_IB,QUA)
%
% Function used for converting quaterions to Euler angles. This function
% also calculates the angular velocity between B and O represented in B.
%
% Input:
% P - struct containg necessary parameter(s) (w_o)
% W_B_IB - Is W^B_IB, a matrix containing the three rotational velocity
% components for all simulation timesteps. Size Nx3
% QUA - Matrix containing quaternions for all simlation timesteps.
% Size Nx4. Form: [eta eps1 eps2 eps3].
%
% Output:
% W_B_OB - Is W^B_OB, a matrix containing the three rotational velocity
% components for all simulation timesteps. Size Nx3
% EUL - Matrix containing Euler angles for all simulation timesteps.
% Size Nx3. Form: [phi, theta, psi].
% Angles are in degrees.
%
% Example:
% [w_B_OB, eul] = qua2eul(P,yout(:,1:3),yout(:,4:7));
%
% Written by Per Kolbjørn Soglo, 1994
% Slightly edited by Zdenko Tudor, 2011
%*****************************************************************

function [w_B_OB,eul] = qua2eul(P,w_B_IB,qua)
% Create angulary velocity vector
w_o=P.w_o;
w_O_IO = [0;-w_o;0];

% Calculating number of vectors
n = size(qua,1);

% Storing inital angle in radians
psi_gammel = 0*pi/180;

% Initiate storing matrices
w_B_OB = zeros(n,3);
eul = zeros(n,3);
for i=1:n,
    % Rotation matrix from O to B
    eta = qua(i,1);
    e1 = qua(i,2);
    e2 = qua(i,3);
    e3 = qua(i,4);
    R11 = eta^2+e1^2-e2^2-e3^2;
    R12 = 2*(e1*e2-eta*e3);
    R13 = 2*(e1*e3+eta*e2);
    R21 = 2*(e1*e2+eta*e3);
    R22 = eta^2-e1^2+e2^2-e3^2;
    R23 = 2*(e2*e3-eta*e1);
    R31 = 2*(e1*e3-eta*e2);
    R32 = 2*(e2*e3+eta*e1);
    R33 = eta^2-e1^2-e2^2+e3^2;
    R_O_B = [R11 R12 R13
             R21 R22 R23
             R31 R32 R33];
    R_B_O = R_O_B';

    % Calculating angular velocity between B and O represented in B
    w_B_OB(i,:) = w_B_IB(i,:)' - R_B_O*w_O_IO;

    % Calculating Euler angles
    psi = atan2(R21,R11);
    if psi < 0
        psi = psi+2*pi;
    end
    psi_test = pi+psi;
    if psi_test > 2*pi
        psi_test = psi_test-2*pi;
    end
    % Checking for discontinuoities in psi
    d = min([abs(psi_gammel-psi), abs(psi_gammel-2*pi-psi), abs(
      psi_gammel+2*pi-psi)]);
    d_test = min([abs(psi_gammel-psi_test), abs(psi_gammel-2*pi-psi_test
      ),...
      abs(psi_gammel+2*pi-psi_test)]);
    if d_test < d
        psi = psi_test;
    end

    theta = atan2(-R31, cos(psi)*R11 + sin(psi)*R21);
    phi = atan2(sin(psi)*R13 - cos(psi)*R23, -sin(psi)*R12 + cos(psi)*
      R22);
    psi_gammel = psi;
    if psi > pi
```

```
            psi = psi - 2*pi;
        end

        eul(i,1) = phi*180/pi;
        eul(i,2) = theta*180/pi;
        eul(i,3) = psi*180/pi;
end
```

```
%*******************************************************************
% File outFcn.m
%
% status = outFcn(t,~,flag)
%
% Is run when ODE45 successfully completes a step.
% This function is used implicitly by ODE45, when ODE settings are
% properly set.
%
% Example:
% options = odeset('OutputFcn',@outFcn,'Refine',1);
% Refine sets how many steps should be created for each successfull step
    .
% With this set to 1 length of all vectors created by the OutputFcn will
    be
% the same as the length of ODE timevector TOUT.
%
% Written by Zdenko Tudor, 2011
%*******************************************************************

function status = outFcn(t,~,flag)
persistent ite;
global tmpVAR VAR;
switch flag
    case 'init'
        %initialize arrays etc.
        ite = 1;
        VAR.moment = zeros(10000,3);
        VAR.torque = zeros(10000,3);
        VAR.J = zeros(10000,1);
        VAR.t = zeros(10000,1);

        status = 0;
    case 'done'
        VAR.moment(ite+1:end,:) = [];
        VAR.torque(ite+1:end,:) = [];
        VAR.J(ite+1:end,:) = [];
        VAR.t(ite+1:end,:) = [];

        status = 0;
    otherwise
        ite = ite+1;
        VAR.moment(ite,:)=tmpVAR.moment';
        VAR.torque(ite,:)=tmpVAR.torque';
        VAR.W(ite,:)=tmpVAR.W;
```

```
            VAR.t(ite,1)=t(end);

            status = 0;
end
end
```

```
%*******************************************************************
% File IGRF11_load_data.m
%
% Section 1:
% Create Matlab variables g_data and h_data containing g and h
% coefficients for the IGRF model from an excel file found online. The
% coefficients stored in matlab variables are compensated with the
  secular
% variation.
%
% Section 2:
% Generate C header file with the IGRF coefficients to be used on a
% microcontroller.
%
% Written by Zdenko Tudor, 2011
%*******************************************************************

[data,txt] = xlsread('IGRF11_data_file.xlsx');

% Input parameters
year = 2011;
day = 53;                   % Day of year

O = 1;
for i=1:length(data)
    col1 = data(i,1);
    col2 = data(i,2);
    col3 = data(i,3);
    col4 = data(i,4);
    if strcmp(txt(i),'g')
        g_data(O+col1,O+col2) = col3;
        g_dataSV(O+col1,O+col2) = col4;
    elseif strcmp(txt(i),'h')
        h_data(O+col1,O+col2) = col3;
        h_dataSV(O+col1,O+col2) = col4;
    end
end

g_data = g_data+((year-2010)+day/365)*g_dataSV;
h_data = h_data+((year-2010)+day/365)*h_dataSV;

save IGRF11_data g_data h_data

%% Generate C header file for IGRF coefficients
filename = 'IGRFcoeffs.h';
```

```

load('IGRF11_data');
fid = fopen(filename,'w');

% Start the header file and include some of the constants
fprintf(fid, 'static const double h_data[14*14] = {+\n');

L = size(h_data,1);
O = 1;
for i=0:L-1
    for j=0:L-1
        fprintf(fid,' %.3f,', h_data(i+O,j+O));
        if mod(i*L+j+1,10) == 0
            fprintf(fid,'+\n');
        end
    end
end
fseek(fid,-1,'eof'); fprintf(fid,''); % Remove last comma character
fprintf(fid,'};\n\n');

fprintf(fid, 'static const double g_data[14*14]={+\n');
for i=0:L-1
    for j=0:L-1
        fprintf(fid,' %.3f,', g_data(i+O,j+O));
        if mod(i*L+j+1,10) == 0
            fprintf(fid,'+\n');
        end
    end
end
fseek(fid,-1,'eof'); fprintf(fid,''); % Remove last comma character
fprintf(fid,'};\n\n');
fprintf(fid, '\n#endif /*_IGRFcoeffs_*/\n');

fclose(fid);
```

C.2 C Code

```
/* File: attitudeControl.c */

#include <avr/io.h>
#include <stdio.h>
#include <math.h>
```

```

#include <util/delay.h>
#include <stdlib.h>

#include "profiling.h"
#include "IGRFcoeffs.h"

#define BAUDRATE 9600                           //set desired baud rate
#define MYUBRR (F_CPU/8/BAUDRATE-1)             //calculate UBRR value

void USART_Init(unsigned int ubrr){
    UBRR0H = (unsigned char)(ubrr>>8);          //low byte
    UBRR0L = (unsigned char)ubrr;               //high byte
    UCSR0B |= (1<<RXEN0)|(1<<TXEN0);            //Enable Transmitter and
      Receiver
    UCSR0A |= (1<<U2X0);
}

void USART_vSendByte_Pc(uint8_t u8Data, FILE *stream){
     while((UCSR0A &(1<<UDRE0)) == 0);          // Wait if a byte is being
      transmitted
    UDR0 = u8Data;                              // Transmit data
}
static FILE mystdout = FDEV_SETUP_STREAM(USART_vSendByte_Pc, NULL,
  _FDEV_SETUP_WRITE);

// Oribt and math constants
static const double pi, Re, Rc;

// Coil and power supply constants
static const double V;
static const double Rz, iz_max, Nz, Az;


/*  Initialize PWM to desired opeartion by modifying its registers */
void PWM_init(){
    // Waveform generation mode: PWM, Phase and Frequency Correct
    TCCR1A |= (0<<WGM11)|(0<<WGM10);
    TCCR1B |= (1<<WGM13)|(0<<WGM12);

    // No prescaling
    TCCR1B |= (0<CS12)|(0<<CS11)|(1<<CS10);

    // Clear on compare match when up-counting. Set on compare match
      when downcounting.
    TCCR1A |= (1<<COM1A1)|(1<<COM1A0);
    TCCR1A |= (1<<COM1B1)|(1<<COM1B0);
    TCCR1A |= (1<<COM1C1)|(1<<COM1C0);

    // No filtering
    TCCR1B |= (0<<ICNC1)|(0<<ICES1);

    // Resolution: 100
    ICR1 = 0x0064;

    // DDRB |= (1<<DDB0)|(1<<DDB4);
    DDRB = 0xFF;
}


/* A function returning the local geomagnetic field (using IGRF-11 model
  ) in orbit frame */
void IGRF(double *out, unsigned short int n, unsigned short int m,
  double lat, double lon){
    unsigned short int i, j;
    double dP[(n+1)*(m+1)+1];
    double P[(n+1)*(m+1)+1];
    double S[(n+1)*(m+1)+1];
    double deltaFun, K;
    double mt, tol;
    double theta, phi;
    double Bt2, Bp2, Br2;
    double Bt1, Bp1, Br1;

    // These arrays need to be initialized as ones
    for(unsigned short int k=0;k<=(n+1)*(m+1);++k)
        P[k] = 1;
    for(unsigned short int k=0;k<=(n+1)*(m+1);++k)
        S[k] = 1;

    Bt2 = 0; Bp2 = 0; Br2 = 0;

    i = n;
    j = m;
    tol = pow(10,-9);

    // Modulation is done in degrees as C does not accept floating point
      modulation
    // (floating point does not HAVE a remainder)
    mt = ((int)(lat*180/pi) % (360));
    mt = mt*pi/180;

    if (mt>((pi/2)-tol) && mt<((pi/2)+tol)){
        lat = (pi/2)+tol;
    }
    else if(mt>((3*pi/2)-tol) && mt<((3*pi/2)+tol)){
        lat = (3*pi/2)+tol;
    }

    theta = (pi/2)-lat;
    phi = lon;

    for (n=0;n<=i;++n){;
        for (m=0;m<=j;++m){
            if (n>0) {
                if (m==0){
                    S[n*(j+1)+m] = S[(n-1)*(j+1)]*(2*n-1)/n;

                }
                else{
                    if (m==1)
                        deltaFun = 1;
                    else
                        deltaFun = 0;
                    S[n*(j+1)+m] = S[n*(j+1)+(m-1)]*sqrt(((n-m+1)*(
                      deltaFun+1))/(n+m));
                }
            }
        }
    }

    for (n=0;n<=i;++n){
        for (m=0;m<=j;++m){
            if (n==1){
                dP[n*(j+1)+m] = cos(theta)*dP[(n-1)*(j+1)+m]-sin(theta)*
                  P[(n-1)*(j+1)+m];
                P[n*(j+1)+m] = cos(theta)*P[(n-1)*(j+1)+m];

            }
            else if (n>1){
                K = (pow(n-1,2)-pow(m,2))/((2*n-1)*(2*n-3));
                dP[n*(j+1)+m] = cos(theta)*dP[(n-1)*(j+1)+m]-sin(theta)*
                  P[(n-1)*(j+1)+m]-K*dP[(n-2)*(j+1)+m];
                P[n*(j+1)+m] = cos(theta)*P[(n-1)*(j+1)+m]-K*P[(n-2)*(j
                  +1)+m];
                }
                if ((n==m) && (n>0)){
                    P[n*(j+1)+m] = sin(theta)*P[(n-1)*(j+1)+(n-1)];
                    dP[n*(j+1)+m] = sin(theta)*dP[(n-1)*(j+1)+(n-1)]+cos(
                        theta)*P[(n-1)*(j+1)+(n-1)];
                }
            }
        }

        for (n=1;n<=i;++n){
            Bt1 = 0;
            Bp1 = 0;
            Br1 = 0;
            for (m=0;m<=j;++m){
                Bt1 = Bt1 + (S[n*(j+1)+m]*g_data[n*(13+1)+m]*cos(m*phi)+S[n
                    *(j+1)+m]*h_data[n*(13+1)+m]*sin(m*phi))*dP[n*(j+1)+m];
                Bp1 = Bp1 + (m*S[n*(j+1)+m]*h_data[n*(13+1)+m]*cos(m*phi)-m*
                    S[n*(j+1)+m]*g_data[n*(13+1)+m]*sin(m*phi))*P[n*(j+1)+m];
                Br1 = Br1 + (S[n*(j+1)+m]*g_data[n*(13+1)+m]*cos(m*phi)+S[n
                    *(j+1)+m]*h_data[n*(13+1)+m]*sin(m*phi))*P[n*(j+1)+m];
            }
            Bt2 = Bt2 + (pow(Re/Rc,n+2)*Bt1);
            Bp2 = Bp2 + (pow(Re/Rc,n+2)*Bp1);
            Br2 = Br2 + (n+1)*(pow(Re/Rc,n+2)*Br1);
        }

        // In Orbit frame, in Cartesian coordinates
        double dummyEps;
        dummyEps = 0;
        out[0] = (Bt2*cos(dummyEps)-Br2*sin(dummyEps))*pow(10,-9);
        out[1] = (-Bp2/sin(theta))*pow(10,-9);
        out[2] = (-Bt2*sin(dummyEps)-Br2*cos(dummyEps))*pow(10,-9);
    }


    /* The desired current from the control law may be larger then the
      maximum current that can run
        through the coil. This function scales the currents so that none
          exceed the maximum possible
        current. Note that all the currents have to be scaled equally to
          preserve the vectors directionallity */
    double currentScaling(double *m_B){
        // NEED: More complex scaling structure needed for multiple coils,
          in order to preserve tau_m's direction
        // Check matlab code for the scaling logic
    double iz;

    iz = m_B[2]/(Nz*Az);
    if (abs(iz) > iz_max){
        if (iz > 0)
            return iz_max;
        else
            return -iz_max;
    }
    else {
        return iz;
    }
}

int main(void){
    PWM_init();
    USART_Init(MYUBRR);
    stdout = &mystdout;

    // Initialize variables variables
    double lat, lon , w_B_IB[3], quaternions[4], eps[3], eta;
    double w_o, detumblingGain, referenceGain;;
    double R_B_O[9], w_B_OB[3];
    double B_O[3], B_B[3], B_B_normSq;
    double m_B[3], iz;

    // Orbit parameters
    w_o = sqrt((6.67428*pow(10,-11)*5.972*pow(10,24))/pow(Rc,3));

    // MATLAB CMD: d = 4e-5;
    detumblingGain = 4*pow(10,-5);
    // MATLAB CMD: k = 2e-8;
    referenceGain = 2*pow(10,-8);

    // Input variables initiated with arbitrary test values. In practice
       these values
    // will be received from other modules in the system, mostly
      attitude determination.
    lat = 0; lon = 0;
    w_B_IB[0] = 0;
    w_B_IB[1] = -w_o-0.2;
    w_B_IB[2] = 0;
    quaternions[0] = 0;
    quaternions[1] = 0;
    quaternions[2] = 0;
    quaternions[3] = 1;

    // Storing input variables. (Eta is the last in the quaternion
      variable from attitude determination module)
    eps[0] = quaternions[0];
    eps[1] = quaternions[1];
    eps[2] = quaternions[2];
    eta = quaternions[3];

    while(1){
        // MATLAB CMD: R_B_O = (eye(3) + 2*eta*S_eps + 2*S_eps^2)'; - (
          note the transpose)
        R_B_O[0] = 1-2*eps[1]*eps[1]-2*eps[2]*eps[2];
        R_B_O[1] = 2*eta*eps[2]+2*eps[0]*eps[1];
        R_B_O[2] = -2*eta*eps[1]+2*eps[0]*eps[2];
        R_B_O[3] = -2*eta*eps[2]+2*eps[0]*eps[1];
        R_B_O[4] = 1-2*eps[0]*eps[0]-2*eps[2]*eps[2];
        R_B_O[5] = 2*eta*eps[0]+2*eps[1]*eps[2];
        R_B_O[6] = 2*eta*eps[1]+2*eps[0]*eps[2];
        R_B_O[7] = -2*eta*eps[0]+2*eps[1]*eps[2];
        R_B_O[8] = 1-2*eps[0]*eps[0]-2*eps[1]*eps[1];

        // MATLAB CMD: w_B_OB = w_B_IB-R_B_O*w_O_IO;
        w_B_OB[0] = w_B_IB[0]+R_B_O[1]*w_o;
        w_B_OB[1] = w_B_IB[1]+R_B_O[4]*w_o;
        w_B_OB[2] = w_B_IB[2]+R_B_O[7]*w_o;

        // Calculate geomagnetic field from IGRF model
        IGRF(B_O,13,13,lat,lon);

        // MATLAB CMD: B_B = R_B_O*B_O;
        B_B[0] = R_B_O[0]*B_O[0]+R_B_O[1]*B_O[1]+R_B_O[2]*B_O[2];
        B_B[1] = R_B_O[3]*B_O[0]+R_B_O[4]*B_O[1]+R_B_O[5]*B_O[2];
        B_B[2] = R_B_O[6]*B_O[0]+R_B_O[7]*B_O[1]+R_B_O[8]*B_O[2];


        // MATLAB CMD: norm(B_B,2)^2;
        B_B_normSq = B_B[0]*B_B[0]+B_B[1]*B_B[1]+B_B[2]*B_B[2];

        // ##DISSIPATIVE DETUMBLING CONTROLLER##
        // MATLAB CMD: m_B = -(d/norm(B_B,2)^2)*cross(B_B,w_B_OB);
        m_B[0] = -(detumblingGain/B_B_normSq)*(-B_B[2]*w_B_OB[1]+B_B[1]*
          w_B_OB[2]);
        m_B[1] = -(detumblingGain/B_B_normSq)*(B_B[2]*w_B_OB[0]-B_B[0]*
          w_B_OB[2]);
        m_B[2] = -(detumblingGain/B_B_normSq)*(-B_B[1]*w_B_OB[0]+B_B[0]*
          w_B_OB[1]);

        // ##REFERENCE CONTROLLER##
        // Mathematically the reference controller can be seen as a
          generalization of the
        // detumbling controller, and can thus be appended to the
          detumbling controller.
        // Truth statement needs to be constructed from some controller-
          switch theory
        if (1==0){
            m_B[0] = m_B[0]-(referenceGain/B_B_normSq)*(-B_B[2]*eps[1]+
              B_B[1]*eps[2]);
            m_B[1] = m_B[1]-(referenceGain/B_B_normSq)*(B_B[2]*eps[0]-
              B_B[0]*eps[2]);
            m_B[2] = m_B[2]-(referenceGain/B_B_normSq)*(-B_B[1]*eps[0]+
              B_B[0]*eps[1]);
        }

        // MATLAB CMD: [m_B,w] = currentScaling(P,m_B);
        iz = currentScaling(m_B);

        // The actual output settings
        // Setting compare PWM register by the relation of desired
          current to the maximum current
        OCR1A = (0x0064)*(0.025/iz_max);
        if (iz > 0)
            PORTB |= (1<<DDB0);
        else
            PORTB |= (0<<DDB0);
    }
}
```

```
/* File: IGRFcoeffs.h */
#ifndef _IGRFcoeffs_
#define _IGRFcoeffs_

static const double pi = 3.1415926535;
static const double Rc = 6971200;
static const double Re = 6371200;

// Coil and power supply constants
static const double V = 5;
static const double Rz = 100;
static const double iz_max = 0.05;
static const double Nz = 784;
static const double Az = 0.08*0.08;

/* These need to be added when all the coils are included
static const double Rx = 100;
static const double ix_max = 0.05;
static const double Nx = 784;
static const double Ax = 0.08*0.08;

static const double Ry = 100;
static const double iy_max = 0.05;
static const double Ny = 784;
static const double Ay = 0.08*0.08;
*/

static const double h_data[14*14] = {+
 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000,+
 0.000, 0.000, 0.000, 0.000, 0.000, 4912.118, 0.000, 0.000, 0.000,
   0.000,+
 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000,
   -2734.040,+
 -590.173, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000,
   0.000,+
 0.000, 0.000, 0.000, -150.651, 248.379, -539.205, 0.000, 0.000, 0.000,
   0.000,+
 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 286.858, -207.535,
   168.523,+
 -310.116, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000,
   0.000,+
 0.000, 45.273, 190.618, -117.069, 4.337, 100.213, 0.000, 0.000, 0.000,
   0.000,+
 0.000, 0.000, 0.000, 0.000, 0.000, -20.915, 41.795, 61.042, -66.873,
   4.016,+
55.473, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, -57.113,+
-20.856, 6.371, 24.785, 6.084, -28.044, -3.171, 0.000, 0.000, 0.000, 0.000,+
0.000, 0.000, 0.000, 10.900, -19.771, 12.473, -16.942, 16.815, 6.985, -10.342,+
2.158, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, -20.500, 11.600, 12.800,+
-7.200, -7.400, 8.000, 2.200, -6.100, 7.000, 0.000, 0.000, 0.000, 0.000,+
0.000, 2.800, -0.100, 4.700, 4.400, -7.200, -1.000, -4.000, -2.000, -2.000,+
-8.300, 0.000, 0.000, 0.000, 0.000, 0.100, 1.700, -0.600, -1.800, 0.900,+
-0.400, -2.500, -1.300, -2.100, -1.900, -1.800, 0.000, 0.000, 0.000, -0.800,+
0.300, 2.200, -2.500, 0.500, 0.600, 0.000, 0.100, 0.300, -0.900, -0.200,+
0.800, 0.000, 0.000, -0.800, 0.300, 1.700, -0.600, -1.200, -0.100, 0.500,+
0.100, 0.500, 0.400, -0.200, -0.500, -0.800};

static const double g_data[14*14]={+
0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000,+
0.000, 0.000, 0.000, 0.000, -29483.445, -1566.775, 0.000, 0.000, 0.000, 0.000,+
0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, -2409.541, 3021.534,+
1671.692, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000,+
0.000, 0.000, 1341.189, -2330.766, 1228.379, 624.924, 0.000, 0.000, 0.000, 0.000,+
0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 910.997, 811.290, 156.408, -352.061,+
87.066, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000,+
-231.673, 357.773, 198.582, -142.002, -161.611, -6.097, 0.000, 0.000, 0.000, 0.000,+
0.000, 0.000, 0.000, 0.000, 72.456, 68.256, 75.656, -139.224, -24.732, 12.871,+
-75.839, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 0.000, 80.629, -75.115,+
-5.387, 46.903, 14.344, 10.515, 0.684, 5.358, 0.000, 0.000, 0.000, 0.000,+
0.000, 0.000, 24.185, 8.315, -15.073, -5.356, -19.644, 11.944, 11.129, -14.673,+
-3.471, 0.000, 0.000, 0.000, 0.000, 0.000, 5.400, 9.400, 3.400,
  -5.300,+
3.100, -12.400, -0.800, 8.400, -8.400, -10.100, 0.000, 0.000, 0.000,
  0.000,+
-2.000, -6.300, 0.900, -1.100, -0.200, 2.500, -0.300, 2.200, 3.100,
  -1.000,+
-2.800, 0.000, 0.000, 0.000, 3.000, -1.500, -2.100, 1.600, -0.500,
  0.500,+
-0.800, 0.400, 1.800, 0.200, 0.800, 3.800, 0.000, 0.000, -2.100,
  -0.200,+
0.300, 1.000, -0.700, 0.900, -0.100, 0.500, -0.400, -0.400, 0.200,
  -0.800,+
0.000, 0.000, -0.200, -0.900, 0.300, 0.400, -0.400, 1.100, -0.300,
  0.800,+
-0.200, 0.400, 0.000, 0.400, -0.300, -0.300};

#endif /*_IGRFcoeffs_*/
```

Bibliography

[1] Allegro Microsystems. A3953SLB-T Data Sheet. `http://www.allegromicro.com/en/Products/Part_Numbers/3953/3953.pdf`.

[2] Atmel. ATMEGA2561v Data Sheet. `http://www.atmel.com/dyn/resources/prod_documents/doc2549.pdf`.

[3] J. Davis. Mathematical Modeling of Earth's Magnetic Field. 2004.

[4] O. Egeland and J. T. Gravdahl. *Modeling and Simulation for Automatic Control*. Trondheim: Marine Cybernetics.

[5] K. M. Fauske. NCUBE Attitude Control. NTNU. 2002.

[6] J. T. Gravdahl. Magnetic Attitude Control for Satellites. 2004.

[7] P. C. Hughes. *Spacecraft Attitude Dynamics*. Mineola, N.Y.: Dover Publications.

[8] International Earth Rotation and Reference Systems Service. IERS conventions 2010 (IERS technical note; no. 36). 2010.

[9] Jan Tommy Gravdahl's students. Collection of small satellite theses. `http://www.itk.ntnu.no/ansatte/Gravdahl_Jan.Tommy/SpaceActivities.html`.

[10] K. L. Jenssen and K. H. Yabar. Development, Implementation and Testing of Two Attitude Estimation Methods for Cube Satellites. 2011.

[11] H. K. Khalil. *Nonlinear Systems, Third Edition*. Prentice-Hall, Upper Saddle River, New Jersey 07458.

[12] R. Kristiansen. Attitude Controll of Mini Satellite. MSc NTNU. 2000.

[13] LPKF Laser and Electronics AG. ProtoMat S62 Tutor Data Tutorial 2.1, English. `http://hci.rwth-aachen.de/tiki-download_wiki_attachment.php?attId=811`.

[14] K. L. Makovec. A Nonlinear Magnetic Controller for Three-Axis Stability of Nanosatellites. 2004.

[15] National Aeronautics and Space Administration Science. `http://science.nasa.gov`.

[16] National Oceanic and Atmospheric Administration. National Geophysical Data Center's IGRF-11 Coefficient table. `http://www.ngdc.noaa.gov/IAGA/vmod/igrf.html`.

[17] National Oceanic and Atmospheric Administration. National Geophysical Data Cente's IGRF Online calculator. `http://www.ngdc.noaa.gov/geomagmodels/IGRFWMM.jsp`.

[18] Power Stream. American Wire Gauge table. `http://www.powerstream.com/Wire_Size.htm`.

[19] Rukstales K.S., Love J.J. The International Geomagnetic Reference Field, 2005: U.S. Geological Survey Scientific Investigations Map 2964, 5 pls., scale 1:30,000,000. `http://pubs.usgs.gov/sim/2007/2964/`.

[20] P. K. Soglo. 3-aksestyring av Gravitasjonsstabilisert Satellitt ved bruk av Magnetspoler. MSc, NTNU. 1994.

[21] M. W. Spong and M. Vidyasagar. *Robot Dynamics and Control*. Hoboken, N.J. : Wiley, 1989.

[22] Thompson Consluting, Inc. Coil Inductance Calculation Techniques. `http://www.thompsonrd.com/induct2.pdf`.