MASSEY UNIVERSITY
ENGINEERING

SCHOOL OF ENGINEERING AND ADVANCED TECHNOLOGY

Engineering Project
Submitted as part requirement for B.Eng (Hons).

De-tumbling System Development of KiwiSAT

Binglun Li (Larry)

2013

SUPERVISORS

a. **Dr. Johan Potgieter**
b. **Dr. Jonathan Henderson**

Abstract

This project addresses the design and development of a module for the Attitude Determination and Control (ADAC) system of a Low Earth Orbit satellite, KiwiSAT. The module is called "De-Tumble Control System" and has the responsibility of automatically slowing KiwiSAT should it be released from the launch vehicle with abnormally fast rotation.

Table of Contents

Abstract 2

Table of Contents...3

Introduction...4

Project Background...5

Literature Review...16

Project Overview 19

Stage 1 – Moments of Inertia Measurement 20

Stage 2 – Sensing Rotation..31

Stage 3 – Control System Development 37

Stage 4 – Simulation 41

Stage 5 – Simulation Results Analysis...43

Stage 6 – Translate Simulated Control System to C...44

Conclusion 45

Reference 46

Appendix 1 – Images..47

Appendix 2 – Simulation Software Code 55

Introduction

The De-Tumble Control System to be developed, has to satisfy a requirement of being able to slow a fast rotating satellite’s motion of 7 RPM to very low, 0.1 RPM.

In order to achieve the aim and satisfy such a requirement, some objectives are set up to be achieved, which would also be called as different stages of the project development.

- 3 Principle Moments of Inertia Measurement
- Control System Concepts Development
- Control System Model
- Control System Simulation
- Translate Simulated Control System to C

After these stages of development, a fully functioning de-tumble control system is expected to be ready for uploading to the actual satellite and used for its mission.

Moreover, the stages mentioned above structure the development of the de-tumble control system in a very manageable manner, which is also how the following report addresses and introduces the details of the development process.

Project Background

The KiwiSAT Project

The KiwiSAT project is undertaken and directed by The Radio Amateur Satellite Corporation of New Zealand (AMSAT- ZL). It is dedicated to design, develop, make, test and operate of the first satellite for New Zealand. It will be a great achievement for New Zealand, since there are few countries other than the US, Russia, the UK, China and Japan capable of building satellites themselves. The satellite is developed and built for communication and science research purposes and the completed model is going to be launched by the Space Launch System, DNEPR, from Yasne in Russia.

KiwiSAT is classified as a Microsat Class amateur communication satellite. After KiwiSAT successfully enters the pre-set orbit, it will become a part of the Orbiting Satellite Carrying Amateur Radio (OSCAR) constellation, with a unique destination, which is allocated by AMSAT-International. Moreover, KiwiSAT would operate under the radio regulations from International Telecommunication Union (ITU) and within the Amateur Satellite Service. (Kennedy, 2011)

In addition, KiwiSAT will be freely open for use by the world-wide Amateur Satellite Community. Members around the world are welcome to use KiwiSAT for communication. Data measured by the on board science package will also be made available on the AMSAT-ZL website or can be directly downloaded from the satellite with the proper receive gear. (Kennedy, 2011)

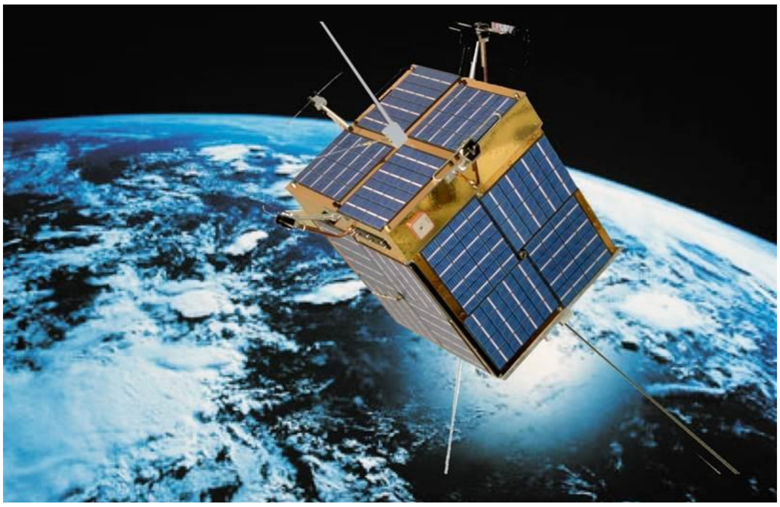


Figure 1 – Animated KiwiSAT Model

The Stakeholders

As mentioned before, KiwiSAT is a project that is undertaken and directed by AMSAT-ZL. Therefore, AMSAT-ZL is the major stakeholder of the project. Moreover, a KiwiSAT Project Team has been formed, with the members from AMSAT-ZL and some other organizations, to conduct the actual research and development work of the satellite. In addition, the KiwiSAT project also has many sponsors from different fields, such as Massey University, AMSAT-International, Fisher and Paykel Healthcare, EADS Astrium and many more. (AMSAT-ZL, 2006)

AMSAT-International

AMSAT is an acronym for Radio Amateur Satellite Corporation. It was founded in 1969 in the US as an educational organization, with the aim of encouraging the amateur radio enthusiasts to conduct more space research. It is a nonprofit making organization of persons who design, build, launch and use small satellites for their own interest and satisfaction. In addition, AMSAT helps to design, build, arrange launches and operate satellites that are carrying amateur radio payloads, such as OSCAR serious of satellites.

Over the years, AMSAT has grown and there are currently 22 countries that have local groups that make up AMSAT-International community. A large portion of these 22 affiliated national organizations have built and launched their own amateur satellite to date. The New Zealand branch of AMSAT is known to be AMSAT-ZL, which has members actively contributing to the global community and the development of new amateur satellite project, such as KiwiSAT.

AMSAT-ZL

As just mentioned, AMSAT-ZL is the New Zealand branch of the AMSAT-International family, which is also the main stakeholder of the KiwiSAT project. It organizes funds, man-power and some other resources for the project. KiwiSAT project team is one of the examples – it organizes volunteer development and electronics engineers, such as Austin Green, Clayton Gumbrell, Terry Osborn, who have contributed significantly to the design, construction and testing of the electronics and control software of the satellite; Fred Kennedy, the project manager and the main driving force of the project, who spent a lot of time and effort in ensuring the success of the KiwiSAT project; Dr. Jon Henderson, Attitude Determination and Control developer, who came up with simple and sophisticated solutions to the orientation control system of the satellite and Ian Ashley, who contributed to the testing and operation of the satellite. All of them have made significant contributions to the KiwiSAT project and should be highly regarded accordingly.

Massey University

Massey University supports the development of the satellite by providing facilities, students and some other resources. For example, computers hardware and software are provided for the KiwiSAT Project Team for development work; a clean-room is provided for the actual satellite's construction and testing; many students are glad to be invited into the KiwiSAT team to work on various of projects, such as calibrating sensors or the orientation control system development.

Figure 2 – Astrium's Logo

EADS Astrium

Astrium is an aerospace subsidiary of European Aeronautic Defense and Space Company (EADS), which provides civil and military space system and services. It is ranked as the 3rd largest surveillance satellite producer in the world. Moreover, Astrium expressed their interests in using KiwiSAT for some science research that related to global warming. It employs an extra 435 MHz UHF beacon to the current satellite's communication package, which will be used to test the advantages of using P band radar for biomass and ice thickness measurement. (Kennedy, 2011)

Sponsors from Industry

There are many companies across many different fields that support KiwiSAT project development progress, for instance, Fisher and Paykel Healthcare, which helps to conduct the vibration test of the satellite. Others are shown below. (AMSAT-ZL, 2006)

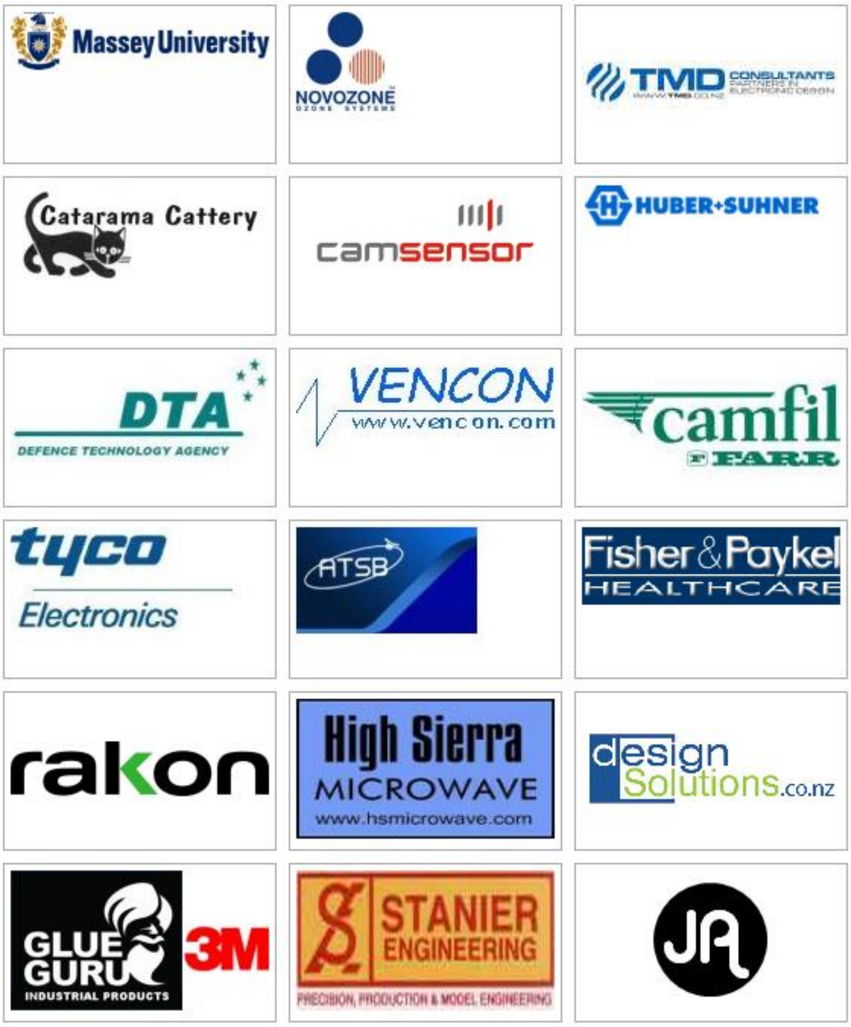


Figure 3 – Sponsors for the KiwiSAT Project

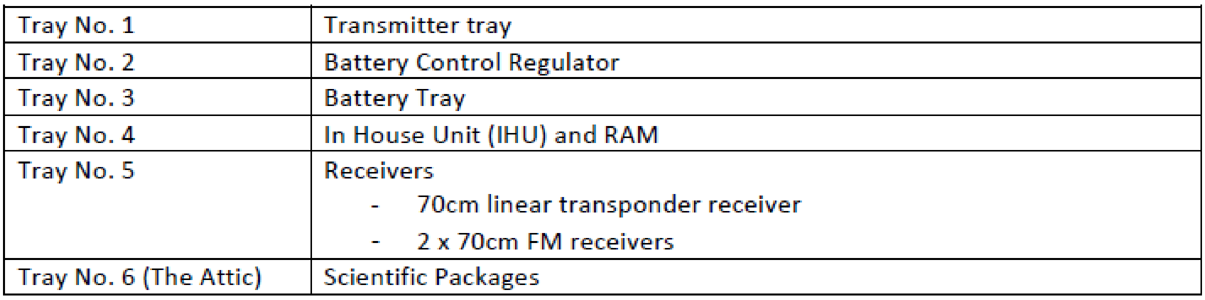
The Satellite

KiwiSAT belongs to “Microsat” class, due to its relatively small size. It is an approximated cube object, 240x240x280mm in size and weighs about 12 kg. The satellite is now in its final stage of development – all the electronics are assembled in place, sensors are calibrated and only waiting for the completion of the control system development.

The satellite will be launched to a Low Earth Orbit, which is approximately 800 to 900 km above the Earth’s surface and expected to take about 100 minutes for the satellite to complete each orbit (travelling in 30,000km/h). Moreover, the satellite will spend about 80% of its time viewing the sun and constantly pass over the North and South poles, as the pre-set orbit is a sun-synchronous orbit, in order to maximize the power generated from the solar cells. (Kennedy, 2011)

In addition, once the satellite is in the orbit, it will become part of the OSCAR (Orbiting Satellite Carrying Amateur Radio) constellation and provide radio services to the worldwide amateur satellite community as its primary functionality. (AMSAT-ZL, 2013)

The satellite is physically built upon 6 aluminum trays that stack on top of another; each of them has been precisely machined and holds the electronics for specific functions or services.



Tray No. 1	Transmitter tray
Tray No. 2	Battery Control Regulator
Tray No. 3	Battery Tray
Tray No. 4	In House Unit (IHU) and RAM
Tray No. 5	Receivers - 70cm linear transponder receiver - 2 x 70cm FM receivers
Tray No. 6 (The Attic)	Scientific Packages

Figure 4 – Table of KiwiSAT Structure

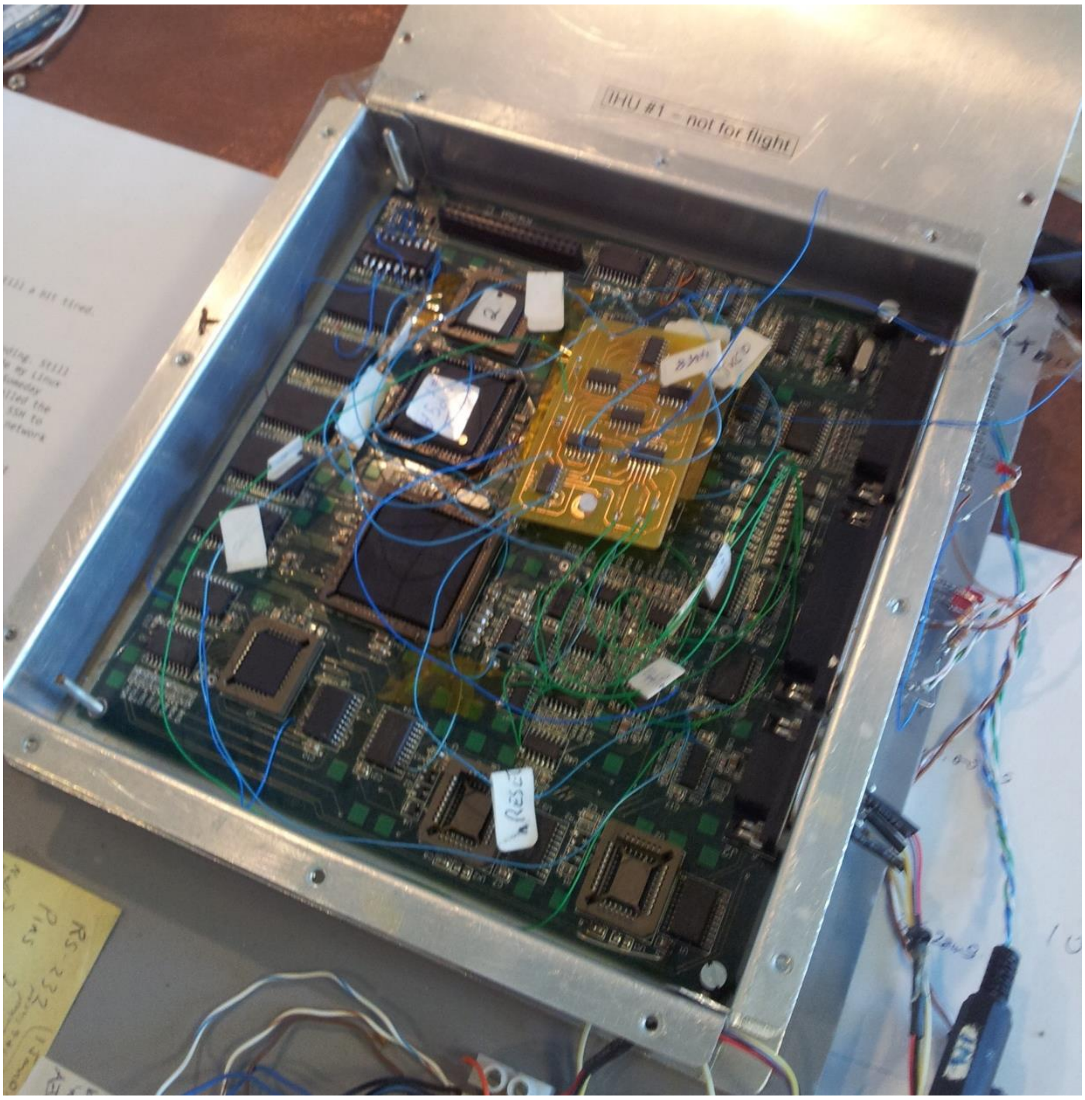


Figure 5 – IHU Tray

The image above is an example for one of the trays, it holds the electronics for the House Keeping unit; more examples will be shown in the appendix.

The exterior of the satellite is covered with solar cells, whose generated electricity will be managed by the Battery Control Regulator and eventually stored in the Battery. As mentioned, the primary function of the satellite is to provide communication services; therefore, the satellite has transmitter and receiver modules built in with antenna around its exterior. These are detailed in the appendix.

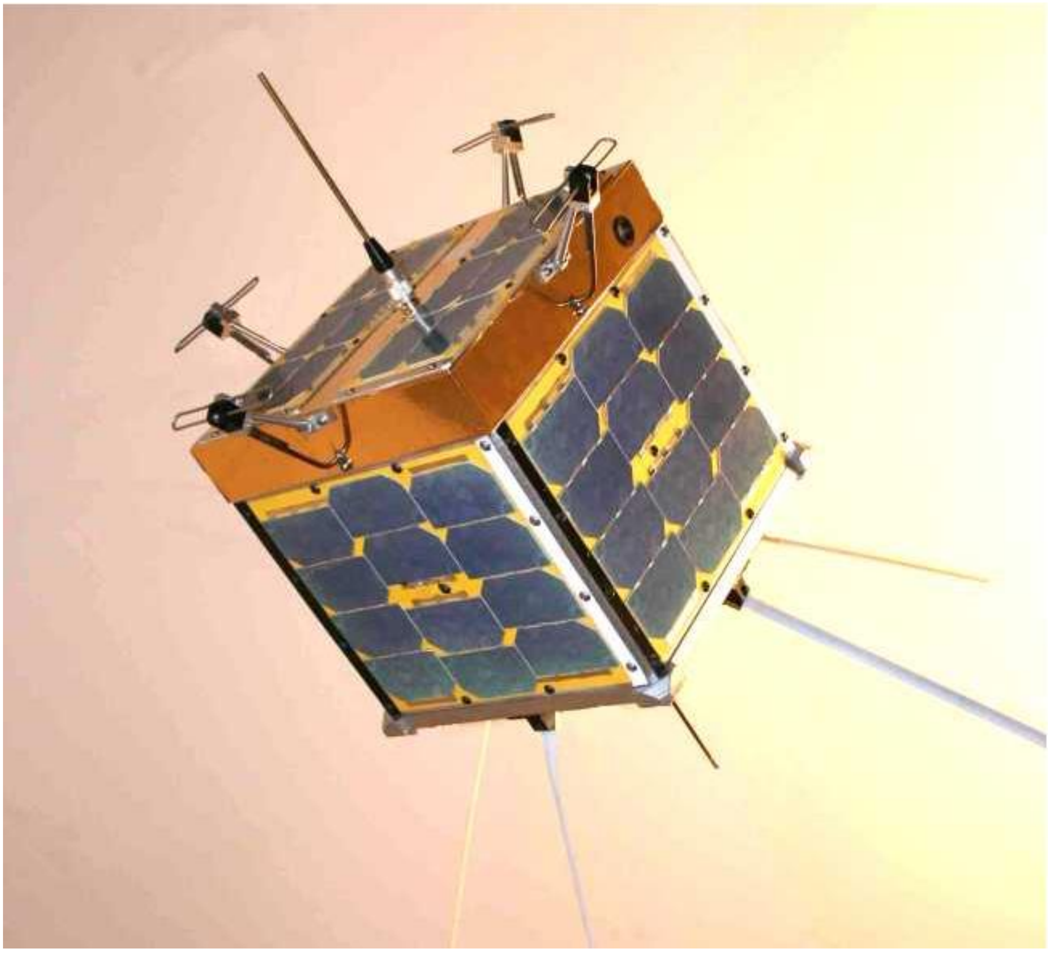


Figure 6 – KiwiSAT Complete Model

Besides communications, the satellite also has a small science package which is designed for the following 2 functions:

1. Ionospheric Characteristics Measurements
2. Attitude Determination and Control Experiments (ADAC)

<u>Ionospheric Characteristics Measurements</u>

Equipment for this experiment was requested by Astrium and uses the 435MHz beacon from KiwiSAT to conduct precise measurements of the ionosphere. Such an experiment would test the advantages of using P band radar for biomass and ice thickness measurement, which is closely related to the global warming issue. (Kennedy, 2011)

Attitude Determination and Control (ADAC)

The ADAC system's core duty is to determine the satellite's attitude, based on the readings from various sensors on board and activate the onboard coils to apply control actions accordingly. Moreover, there are 2 modules inside the ADAC system.

- De-tumble System
- Stable State Orientation Control System

Firstly, the De-tumble System is designed to be used just after KiwiSAT is released from the launch vehicle, in order to stabilize a possibly fast rotating and unstable motion of the satellite. The following report will describe the development of this system in details.

Secondly, when the satellite is in a stable condition, then the second orientation control system would kick in, to do accurate attitude control. Such an orientation control system is very important to the satellite, since it can:

a) prevent one side of the satellite over expose to the sun, which might cause overheating and damage to the solar cells;
b) increase power generation efficiency from the solar cells – actively control and maximize the amount of cells expose to the sun;
c) conduct science research by accurately pointing the satellite to a certain direction and then take measurements from the sensors.

The development of the second orientation control system is another complete project on its own and would not be discussed in the following report.

To achieve the ADAC functionality the science package of KiwiSAT includes a list of sensors, such as:

- 1 three-axis Magnetometer
- 2 Sun Sensors
- 2 pairs of horizon sensors
- CMOS camera
- GPS receiver
- Solar cells

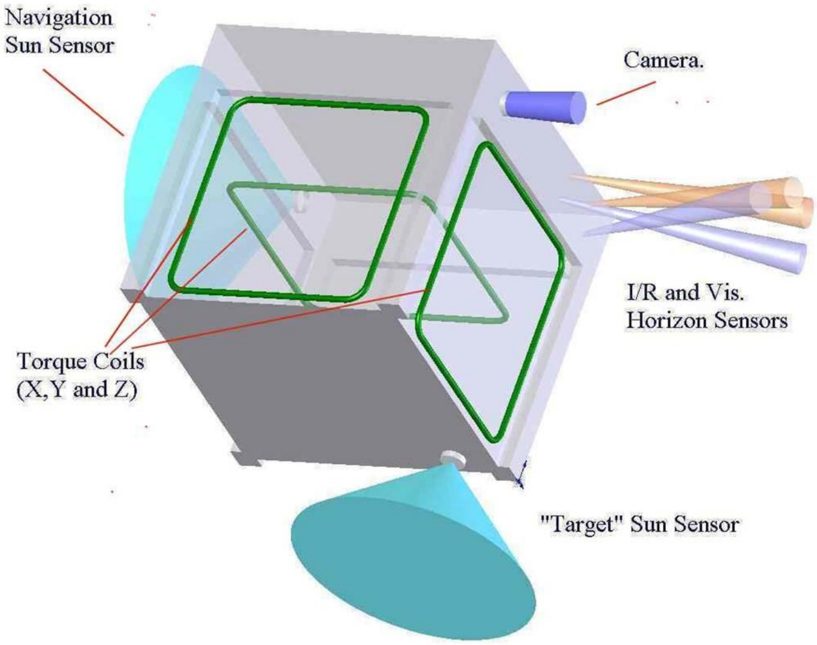


Figure 7 – Sensors and Actuators Onboard

For orientation control, the satellite employs 3 coils – 1 in each axis. All of those coils are identical to each other; they size about 200x200mm, draw 0.5A and generate 1.554 Am^2 when they are powered.

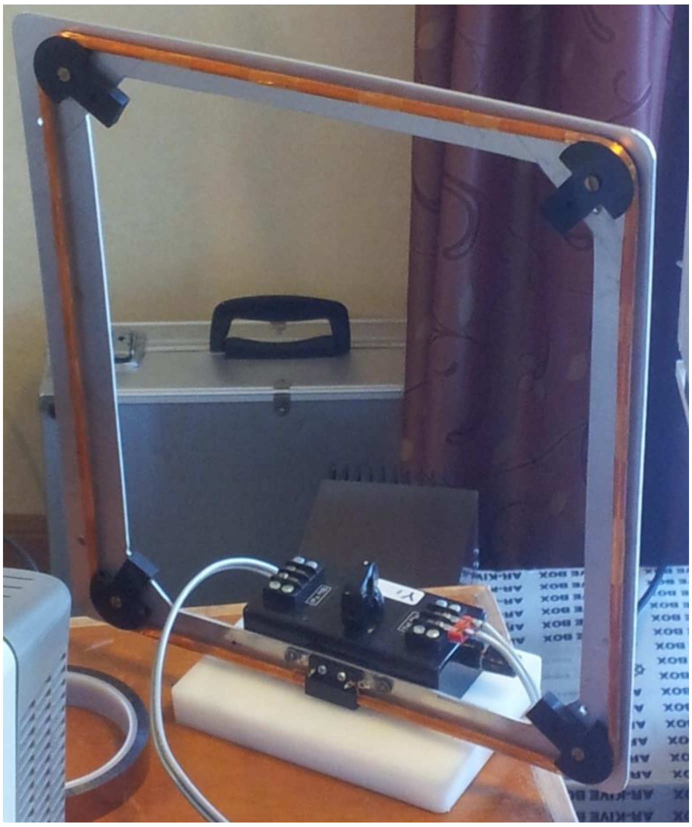


Figure 8 – Onboard Magnetic Coils

When a coil is energized, it will generate a magnetic moment which will react (attract/repel) with the Earth’s magnetic field and yield a torque that is able to change the motion of the satellite. (See Literature Review for more details)

Launch System

Due to the exterior layout of KiwiSAT the DNEPR launch vehicle from Russia is chosen to launch the satellite. Their release system is a gentle way of launching satellites.

Figure 9 – DNEPR Logo

As shown in the image, when the third stage of the rocket approaches to the right position, it would actually turn around and gently release (“dropping”) the satellites while moving forward, which is ideal for KiwiSAT and different from the common “kick in the bottom” method that NASA and European Space Agency use. (Kosmotras , 2001)

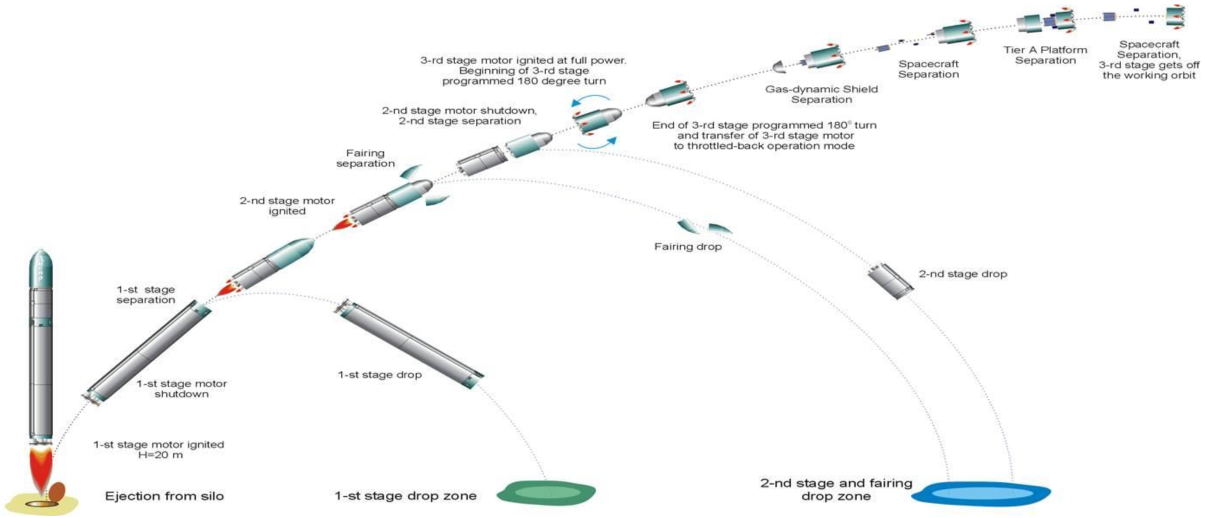

Spacecraft Separation, 3-rd stage gets off the working orbit
Tier A Platform Separation
Spacecraft Separation
Gas-dynamic Shield Separation
3-rd stage motor ignited at full power. Beginning of 3-rd stage programmed 180 degree turn
End of 3-rd stage programmed 180° turn and transfer of 3-rd stage motor to throttled-back operation mode
2-nd stage motor shutdown, 2-nd stage separation
Fairing separation
2-nd stage motor ignited
Fairing drop
2-nd stage drop
1-st stage separation
1-st stage drop
1-st stage motor shutdown
1-st stage motor ignited H=20 m
Ejection from silo
1-st stage drop zone
2-nd stage and fairing drop zone

Figure 10 – DNEPR Launch System

Literature Review

Magnetometer

There is a 3-axis prescient magnetometer installed in the satellite. Magnetometer returns readings that describe the Earth magnetic field's direction and magnitude, respect to the magnetometer itself. (AMSAT-ZL, 2013)

Therefore, if the Earth magnetic field is stationary and the satellite is rotating, then what the magnetometer "sees" is actually the satellite is stationary and the Earth's field is rotating around it with reversed direction.

Theory of Rotation

As the fundamental equation of rigid body dynamics indicated, attitude dynamics relates the time derivative of the angular momentum vector, d**L**/dt, to the applied torque, **T**.

$$\frac{d\boldsymbol{L}}{dt} = \boldsymbol{T} - \boldsymbol{\omega}\boldsymbol{L} = \boldsymbol{I}\frac{d\boldsymbol{\omega}}{dt}$$

Where **L** is the angular momentum vector along the spacecraft-fixed axes; **ω** is the instantaneous velocity vector; **I** is the moments of inertia and **T** is the applied torque, defined below.

$$T = \sum_{i=1}^{n} r_i \times F_i$$

Moreover, the force, **F**, on the *i*th mass consists of 2 parts – externally applied force and an internal force exerted by the other masses.

In addition, by substituting the classic momentum equation,

$$\boldsymbol{L} = \boldsymbol{I}\boldsymbol{\omega}$$

the rigid body dynamics equation could be re-written as,

$$\boldsymbol{I}\frac{d\boldsymbol{\omega}}{dt} = \boldsymbol{T} - \boldsymbol{\omega} \times (\boldsymbol{I\omega})$$

which, could be further simplified to

$$I_1\frac{d\omega_1}{dt} = T_1 + (I_2 - I_3)\omega_2\omega_3$$

$$I_2\frac{d\omega_2}{dt} = T_2 + (I_3 - I_1)\omega_1\omega_3$$

$$I_3\frac{d\omega_3}{dt} = T_3 + (I_1 - I_2)\omega_1\omega_2$$

These are the Euler Equations and it is important to note that **T** & $\boldsymbol{\omega}$ are relative to the satellite axes. (Wertz, 1978)

Current State of Art

There are many methods to actively control the spacecraft attitude concurrently. The most common techniques are mass expulsion; momentum wheels and electromagnetic coils. (Wertz, 1978)

Mass Expulsion System

Mass expulsion control systems, used for attitude control, can be normally found as gas and ion thrusters.

They are simple to operate, efficient in the execution for reorientation and not limited to the space environment. However, they are expensive, require complicated hardware and engine and have limited lifetime by the amount of fuel onboard. Moreover, they can also cause orbit changes when a reorientation is in execution; therefore, thrusters are normally fired in pairs to minimize the side effect of translation motion. (Tate, 1978)

Momentum Wheels

Momentum wheels are used to absorb disturbance torques. They are normally installed on 1, 2 and 3 axes to control the pitch, yaw and roll of the spacecraft.

The operation of the momentum wheel is complex and relies on the interaction of mechanical parts, which have limited lifespan. Moreover, secondary active attitude control system is normally required to maintain the wheels and spacecraft momentum or even used to restore the momentum wheels back to its nominal operation condition. (Tate, 1978)

Momentum wheels system maintains and controls the attitude, by exchanging the momentum between the spacecraft and the wheels. Externally applied torque will be absorbed by the wheels to maintain the spacecraft attitude; on the other hand, controlling the attitude of the spacecraft can be achieved by changing the wheels' momentum.

Magnetic Coils

Magnetic coils control system can be used to control the satellite orientation as far as the orbit altitude is less than 35,000 km. Such a system is relatively lightweight and requires no moving parts or complex hardware. However, it requires relatively large amount of power; provides slow reorientation and its operation depends on the Earth's field configuration. (Tate, 1978)

There are 3 types of magnetic torque system being used – permanent magnets, "air"-core coils and iron-core coils. Permanent magnets are the heaviest type and are used only for stabilization. "Air"-core and iron-core coil systems are used for both stabilization and reorientation.

Comment

Due to many limitations of the satellite, such as size, budget, durability, etc., the "air"-core coils system is chosen to perform the orientation control for KiwiSAT. Moreover, the de-tumble controller, that is about to be developed, is also going to be built upon such an actuating system with some other onboard sensors.

Magnetic-torque

There are 3 magnetic coils, 1 in each axis, in the satellite. As just mentioned, when they are energized, magnetic moments will be generated, which interacts (repel/attract) with the Earth magnetic field and resulting a torque that could change the motion of the satellite.

It is important to note that only the magnetic dipoles that are in the rotational plane are able to control the rotational speed; otherwise the magnetic moments would contribute to making side effects and adding speed to other axes. (Wertz, 1978)

Project Overview

Despite the Kosmotras DNEPR launch company states that the released satellites will be in a stable and slow rotating motion – about 0.5 degree/sec for yaw, pitch and roll; that is not always the case from experiences. In fact, it is suggested that the satellite may be rotating in a relatively high speed – from 2 and even up to 7RPM. If that is the case, it would be very hard to command the satellite to do any desired task, such as reorienting the satellite to a certain direction to take photos or conduct experiments, etc. – such a state is considered to be an unstable.

Therefore, a de-tumble control system has to be in place to automatically stabilize the post-released KiwiSAT. The control system would get readings from sensor(s) and actuate magnetic coils; and it has to be so simple that even the weak onboard CPU can cope with. Besides, it still needs to perform well and be able to slow a high rotation speed of up to 7 RPM down to very low, 0.1 RPM.

Stage 1 – Moments of Inertia Measurement

Reasons from Rotational Theory

As the Euler Equations of Dynamic Motion pointed out, if a torque is applied to an object and want to find out the resulting angular velocity, then the moments of inertia of the object are required. (Wertz, 1978)

$$I_1 \frac{d\omega_1}{dt} = T_1 + (I_2 - I_3)\omega_2\omega_3$$

$$I_2 \frac{d\omega_2}{dt} = T_2 + (I_3 - I_1)\omega_1\omega_3$$

$$I_3 \frac{d\omega_3}{dt} = T_3 + (I_1 - I_2)\omega_1\omega_2$$

Since the actual satellite is not a solid object with unevenly distributed weight, its moments of inertia will be different to each other and cannot be obtained just by calculation from equations. Therefore, some kinds of test method have to be developed, in order to find out the actual moment of inertia of the satellite in 3 different axes.

Testing Method

"Moments of Inertia" simply means the inertia, in the rotational motion domain in 3 different axes of the object. Therefore, they should be measured while the object is rotating. As a result, the testing object is turned into a torsional pendulum for the experiment.

Afterwards, diagrams of the torsional pendulum system are analyzed the mathematical models of the system are obtained.

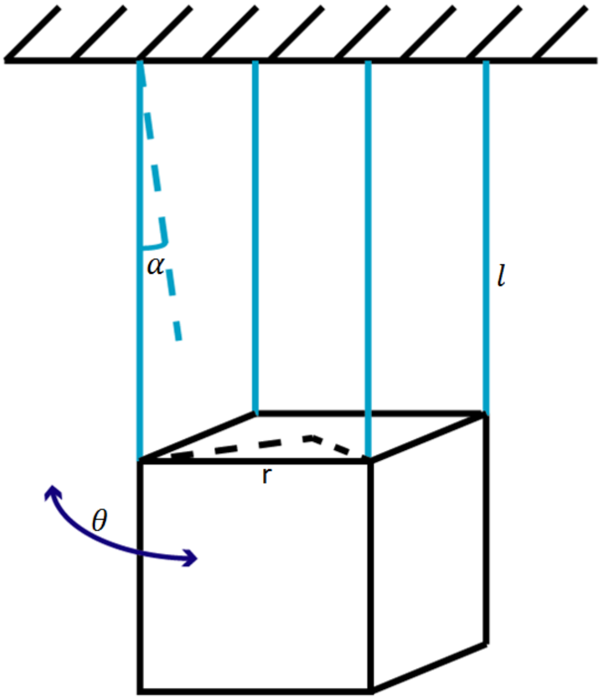


Figure 11 – Torsional Pendulum

The diagram above shows the construction of a simple torsional pendulum – a cube object is hanged from the ceiling using wires with the same length, l. After the object is completely settled and stationary relative to the supporting frame, which is called initial position, the object is rotated with a horizontal offset angle θ/2. After it is released from the offset position, the object will rotate/oscillate around its initial position with a range of θ degree.

$$T = 2\pi\sqrt{\frac{I}{k}}$$

As the classic equation of pendulum indicated, the time of each period is controlled by

a) the moment of inertial of the object, I, and
b) the torsional constant, K, of the test equipment setting, which is dependent on the wire length, L, and the object size, r, etc..

Therefore, In order to find out the moment of inertia using the classic pendulum equation, time for each period (T) and the torsional constant (k) are required. On the other hand, time for each period (T) could simply be measured in the experiment by using a stop-watch; the torsional constancy, however, is really unknown.

Finding out Torsional Constancy

Therefore, more analyses are undertaken to calculate the torsional constancy of the test system.

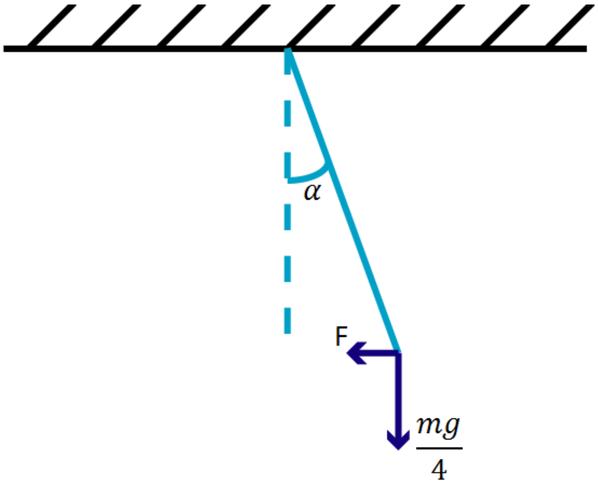


Figure 12 – A Corner of the Torsional Pendulum

Firstly, the view is zoomed into only 1 of the 4 supporting points on the object, to analysis the force that the wire is undertaken (T). It contains 2 force components:

a) firstly, a vertical upward component that overcomes the gravity (mg/4)
b) secondly, a horizontal component (F) always points to the initial position.

The magnitude of the horizontal component F could be calculated from the basic trigonometry equation, with the angle α, offset angle between the initial wire position and the current wire position:

$$F = \frac{Mg}{4}\tan\alpha$$

Secondly, the resulting moment (torque) due to rotation could be obtained by analyzing the top view.

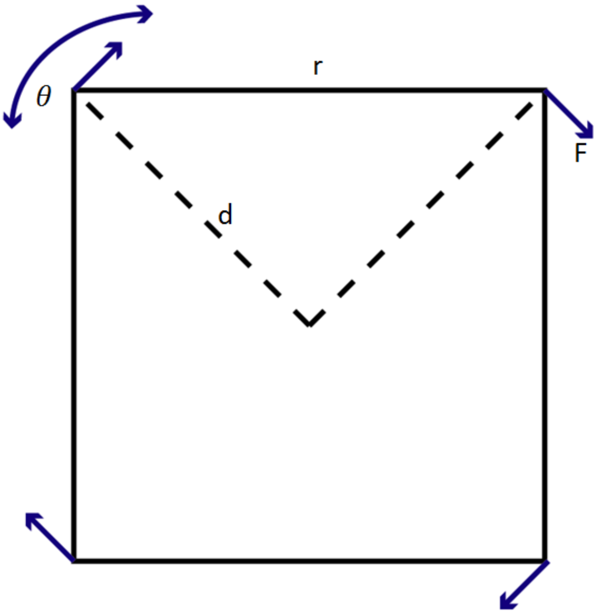


Figure 13 – Top View of the Torsional Pendulum

All 4 of the supporting points experience the same horizontal force F, which had been discussed before, with distance, d, away from the center point. Therefore, the resulting moment could be calculated accordingly.

$$d = \frac{r}{\sqrt{2}}$$

$$M_0 = F \cdot d$$

$$M_0 = \frac{Mg}{4} \tan\alpha \frac{r}{\sqrt{2}} 4$$

In addition, assuming the angle α is very small, therefore:

$$\sin\alpha = \alpha$$

$$\cos\alpha = 1$$

$$\tan\alpha = \alpha$$

Therefore the moment could be further simplified to:

$$M_0 = \frac{Mg}{4}\frac{r}{\sqrt{2}}\alpha \cdot 4$$

On the other hand, α could be linked to θ by another analysis:

$$\alpha l = \theta d$$

$$\alpha l = \frac{r}{\sqrt{2}}\theta$$

$$\alpha = \frac{r}{\sqrt{2}}\frac{\theta}{l}$$

Lastly, after substituting α into the moment equation, a model equation could be obtained:

$$M_0 = \frac{Mg}{2}\frac{r^2}{l}\theta$$

Such an equation simply means that the resulting moment is proportional to the rotational angle θ, where the torsional constancy (K) will be:

$$k = \frac{Mg}{2}\frac{r^2}{l}$$

Rearranging the Classic Pendulum Equation

$$T = 2\pi\sqrt{\frac{I}{k}}$$

Armed with a known torsional constant (k) and a measurable period time (T), moment of inertia of the object (I) could be calculated by rearranging the classic pendulum equation.

$$T^2 = 4\pi^2 I \frac{2l}{Mgr^2}$$

Moreover, the "object", that is actually being tested, consists of two parts – firstly, the object that is being measured and, secondly, the supporting material. Both of the testing object and the supporting material have mass and moment of inertia, therefore the equation above could be further simplified to:

$$T^2 = \frac{8\pi^2 l}{gr^2}\left[\frac{I_s + I_0}{M_s + M_0}\right]$$

Where "s" stands for supporting material and "o" stands for the object that is being tested.

Therefore, the moment of inertia could be described as:

$$I_o = \ CT^2(M_s + M_0) - I_s$$

Where

$$C = \frac{gr^2}{8\pi^2 l}$$

Lastly, the mathematical model could be further simplified to a linear equation

$$I_0 = CX_0 - I_s$$

By grouping

$$X_0 = T^2(M_s + M_0)$$

It is a very impressive mathematical model – despite all of the complex operation, such as square root and square, the moment of inertia could be calculated by using just 1 simple linear equation.

Construction of the Testing Stand

As mentioned before, moments of inertia of the satellite will be measured by turning the satellite into a torsional pendulum. Therefore, a testing stand and some supporting materials have to be designed and built to hang them for the experiment.

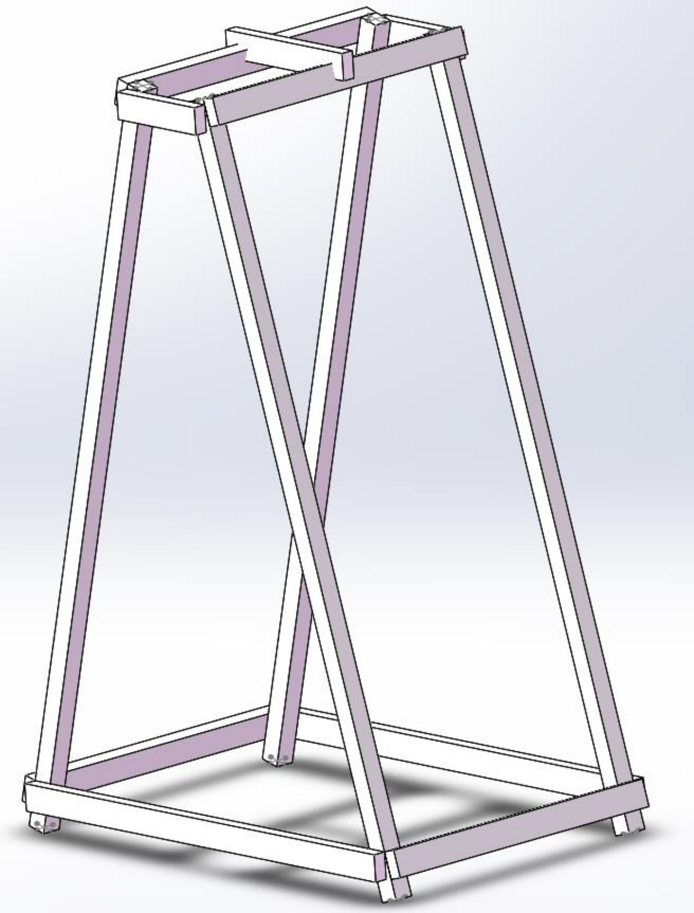


Figure 14 – Testing Stand Designed in SolidWorks

As shown in the diagram, the stand is designed in a pyramid shape and it is about 2.8m tall from the ground. It has a very wide base of about 1.9m and a long side section of 1.3m to gain extra stability as well as to prevent the collisions between the antennas on KiwiSAT and the body of the stand.

Moreover, the stands would hang the satellite 2.5 mete down from the supporting material on the top, which makes the angle α small, so that tan(α) could be approximated to α.

Afterwards, timber is processed into the right lengths and shapes and assembled together with wood screws. Lastly, the testing stand is constructed. As shown in the image below, the stand looks about twice as high as an adult.

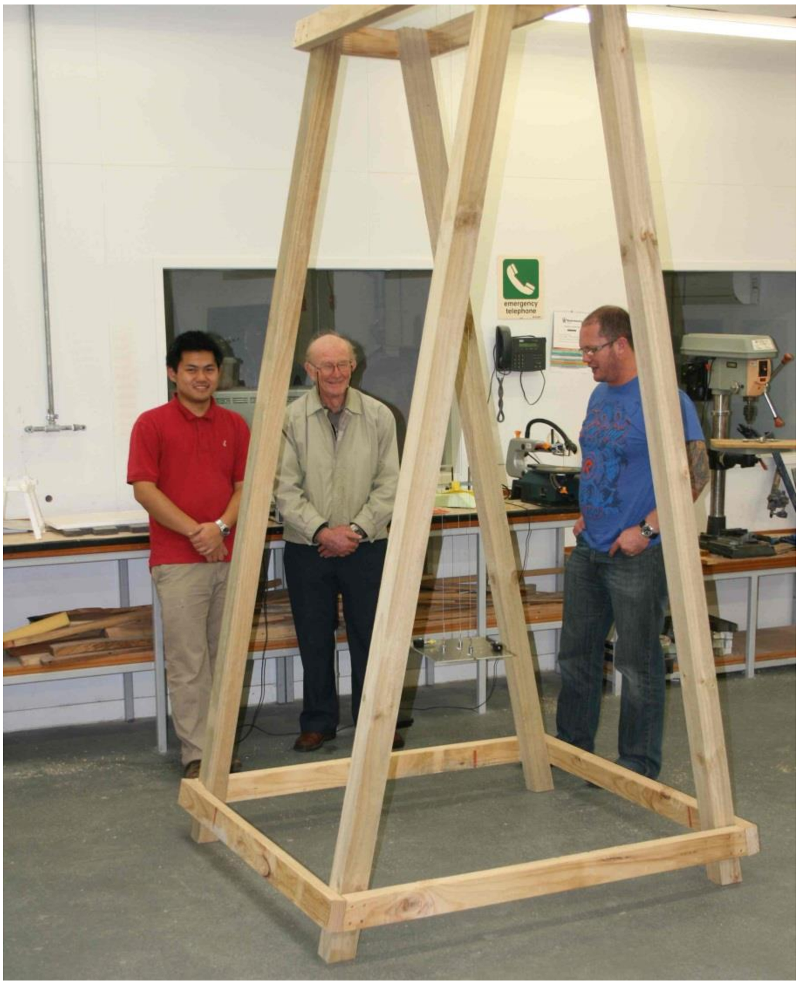


Figure 15 – Fully Constructed Testing Stand

Calibrations and Experiments

As the mathematical model obtained from theory indicated, the moment of inertia of a particular object is in linear relationship with a variable “X_o”, where “C” is the slope and “I_s” is the offset.

$$I_0 = CX_0 - I_s$$

The slope “C” is depended upon the testing equipment, such as, the number of supporting points, the length of wire, the distance between each supporting points, etc. On the other hand, the offset “I_s” is the moment of inertia of the supporting material. Therefore, the slope “C” and offset “I_s” should stay constant, no matter what object is being tested, as far as the testing equipment remains the same. Consequently, the data points plotted on the X_o and I_o diagram should fit on a straight line.

In another word, the slope “C” and the offset “I_s” could be calibrated (or measured) by using the system to test objects with known “I_o” and “X_o”.

Calibration

Three objects are purposely made for calibrating the testing equipment and calculating the slope “C” and offset “I_s”:

a) a big wooden cube, sizes about 300x300x340mm and weights about 12kg;
b) a small piece of timber panel, size about 250x250x40mm and weighs about 1.5 kg;
c) a piece of long timber, sizes about 650x120x40mm and weighs about 1.7kg.

Firstly, the big wooden cube is tested. It has evenly distributed mass and regulator shape; therefore its moment of inertia could be calculated by using another classic equation.

$$I_0 = \frac{1}{12} M_0 (W_1^2 + W_2^2)$$

Secondly, the variable in the equations, X_o, is shown below, combining the square of the time of each period (T), which is measured by using a stopwatch in the experiment, and the sum of mass of the testing object (M_o) and supporting material (M_s).

$$X_0 = T^2(M_s + M_0)$$

Therefore, "I_o" and "X_o" of the big cube are obtained after these 2 steps.

Following the same procedure, the small timber panel and the long timber beam are tested individually.

After all, 3 sets of data are obtained, which are "I_o" and "X_o" values of each individual calibration object.

Results Analysis

Those 3 sets of data are plotted in a graph, with variable "X_o" on the x axis and the "I_o" on the y axis.

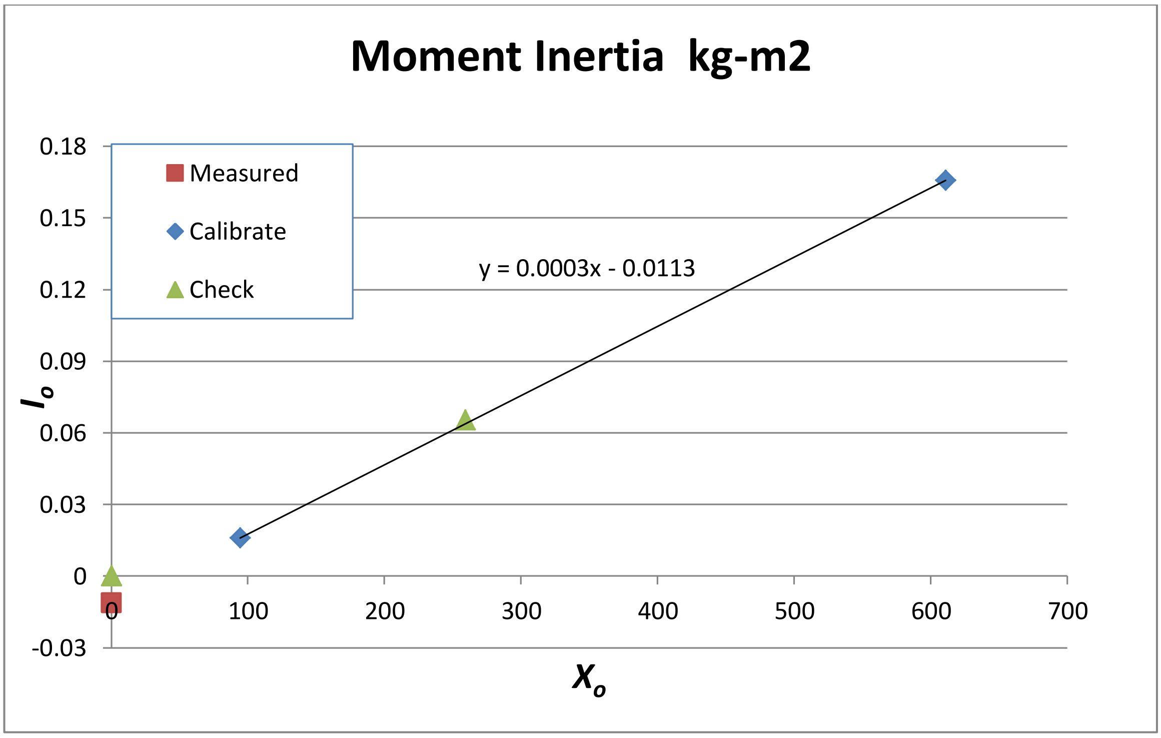


Figure 16 – Moment of Inertia Calibration Results

As shown in the graph, all of those 3 points are highly correlated to each other in a linear pattern, which is proven by an extremely high R-Value, 0.999943.

$$I_0 = CX_0 - I_s$$

$$X_0 = T^2(M_s + M_0)$$

This is very impressive and suggests that the mathematical model obtained from the theory is correct; secondly, the slope "*C*" and the offset "I_s" are calculated from the line to be 0.0003 and 0.0113 respectively.

Summary

In summary, an easy-to-use moment of inertia testing system is fully developed with proper theoretical model. The design has been built and calibrated for the slope "*C*" and offset "I_s". Now the testing equipment is ready to be used to measure the moments of inertia of the actual satellite.

Stage 2 – Sensing Rotation

Rotation Measurement Concepts

In order to control (or stabilize) the post-released satellite rotation, as stated in the project aim, it is essential to be able to sense how the satellite rotate – being able to know how fast the satellite is spinning in what direction.

Consequently, members in the KiwiSAT Project Team stared brainstorming different concepts about how to use the onboard sensors to measure the rotation.

Concept 1 – Absolute Certain Measurement

This concept would measure the rotation of KiwiSAT by constantly trying to calculate the exact orientation of the satellite respective to the Earth. The difference between any 2 attitude-measurements/calculations is caused by the rotation; therefore, the rotational axis, speed and direction could be found out accordingly.

It would, firstly, try to take readings from the magnetometer and both of the sun sensors. Secondly, based on the readings, it calculates the exact orientation of KiwiSAT. Lastly, it calculates the differences between any 2 calculated results to obtain the information of the rotation.

Comment

Although this method seems very logical, it is very unpractical for this application.

Firstly, the exactly orientation of the satellite could be calculated, only when both magnetometer and either of the sun sensor are able to return readings. However, that is not always the case, since there is a significant portion of opportunity that neither of the sun sensors is able to “see” the sun and resulting insufficient reading for the controller to calculate the exact attitude of the satellite. In another word, such a measurement system relies on windows of opportunity that either sun sensors faces towards the sun, whose sizes are absolutely uncontrollable. Secondly, even though the control system is able to get both of the magnetometer and sun sensor readings, it requires very complicated mathematical models to calculate the actual satellite attitude respect to the Earth, which requires far more computational power and memory than the actual on board computer has. By the time it finishes the first attitude calculation, the fast spinning satellite might have already completed a few rotational cycles, which means it would have big aliasing issue.

For all of these mentioned disadvantages, such a measurement method is abandoned.

Concept 2 – Relative Measurement

In this concept, reading form only one sensor, magnetometer, is required to measure the rotation of the satellite. It assumes the Earth magnetic field is constant and therefore, difference between any 2 magnetometer readings is caused by the rotation.

Comment

It is very impressive that how this method is. Therefore, there are a lot of advantages of this concept comparing to the previous one.

Firstly, magnetometer reading is available all the time; the measurement system is not limited to anything that is uncontrollable at all – the window of opportunity of taking reading is always open in this case. Secondly, it is very easy to use, does not need complicated mathematical models and consequently requires far less computational power – any change on the magnetometer readings over the certain time step is due to rotation. As a combined result, aliasing issues could be avoided.

Therefore, the second concept is chosen as the method to measure the rotation.

Measure the Rotation

There are only two parameters of rotation are required for the control system.

a) Speed of the rotation.
b) The inclination angle between the Earth's field and the rotational axis.

Measure Speed

Speed of rotation is required for obvious reason – need to know how fast the satellite is rotating, so that the following issues could be decided:

a) whether or not turning on the de-tumble system;
b) the step size of the control action (will discussed in details latter);

How to achieve it?

As discussed before, magnetometer returns readings that represent the earth's field direction and magnitude, respect to the satellite itself. Moreover, the measurement system assumes the earth's field is constant and takes reading from the magnetometer at a constant time step.

For example, as seen from graph, at time instance 1, the system takes a reading from the magnetometer, yielding B1; after a time step, another reading form the magnetometer is taken, yielding B2; lastly, B3 is also obtained after the same time step by the magnetometer.

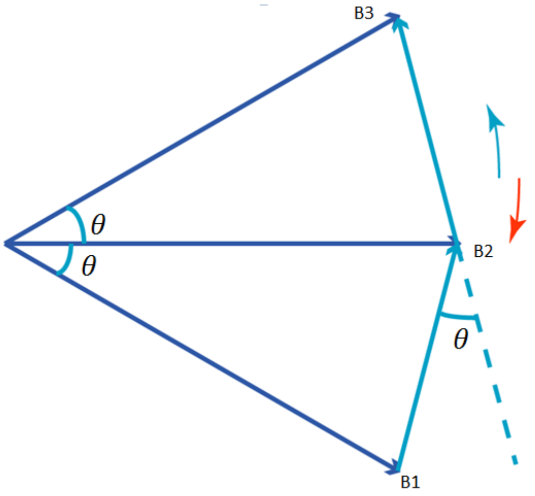


Figure 17 – Rotation Analysis Diagram

Due to the nature of magnetometer, the satellite looks stationary; the earth's field, on the other hand, looks like rotating in an anti-clockwise manner to the satellite in this case.

The angle θ, defined as the rotational angle between 2 measurements, can be calculated in the following equation.

$$\cos\theta = \frac{dot(b_2 - b_1, b_3 - b_2)}{|b_2 - b_1| \cdot |b_3 - b_2|}$$

In another word, θ is the angle that the satellite rotates over a time step; therefore, the speed of rotation can consequently be calculated as θ divided by step-time.

Measure Inclination Angle of the Earth's Field

The inclination angle, α, is defined as the angle between the satellite's rotational axis and the earth magnetic field.

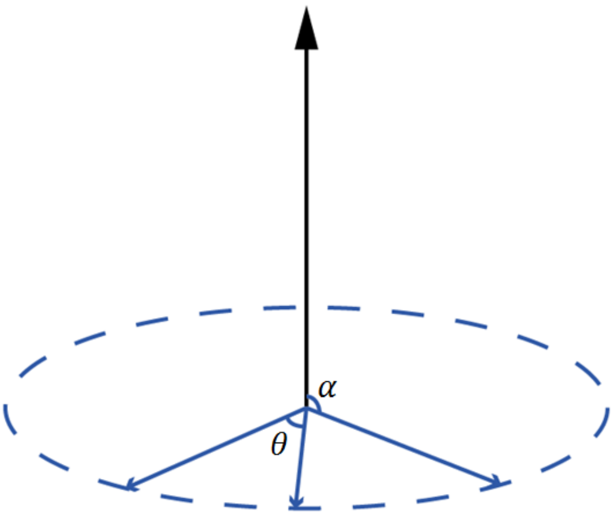


Figure 18 – Different Manners of Rotation

As shown in the images above, the earth's magnetic field would rotate around the satellite (from the magnetometer view) in many different manners – the angle, α, could be anything between 0 degree, parallel to the rotational axis, to 90 degree, perpendicular to the rotational axis.

Measuring the inclination angle is very important, since it will significantly affect the efficiency of magnetic-torque actuation, which will be discussed in the latter chapter.

How to achieve it?

Some equations are obtained after analyzing the inclination diagram.

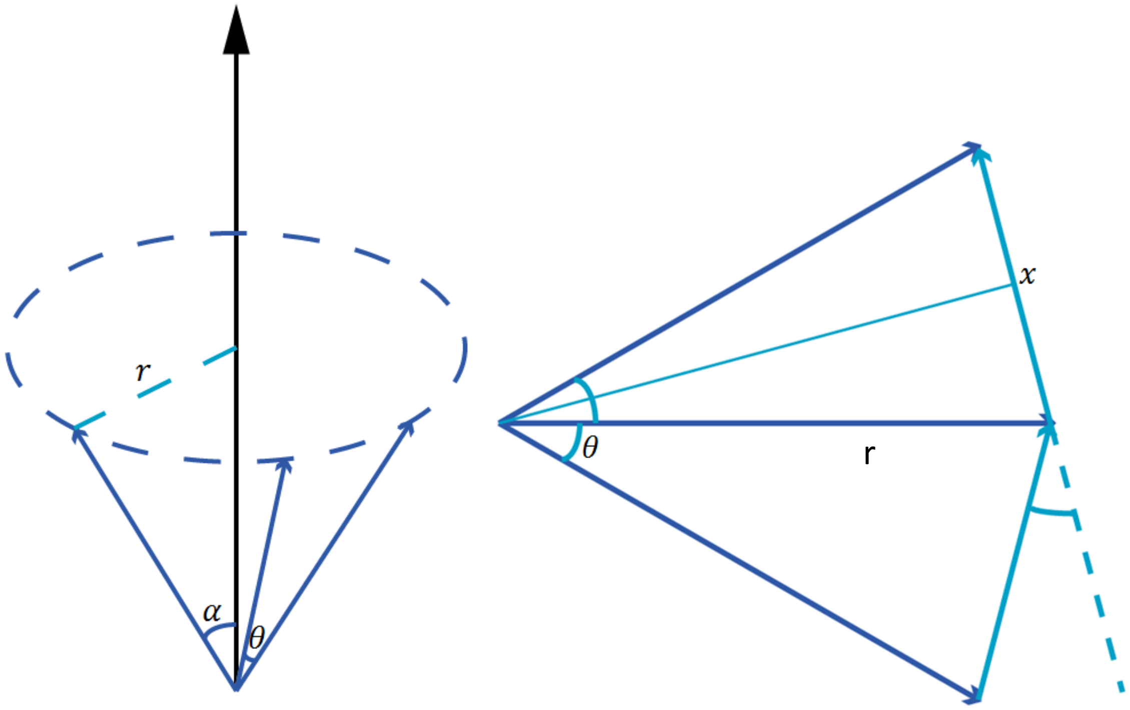


Figure 19 – 3D View of Rotation

Assuming the magnetometer reading has already normalized,

$$r = \sin \alpha$$

$$\sin \frac{\theta}{2} = \frac{\frac{x}{2}}{\sin \alpha}$$

Since θ is already known from the previous step, therefore α could simply be calculated as:

$$\sin \alpha = \frac{\frac{x}{2}}{\sin \frac{\theta}{2}}$$

Summary

In short, a method has been developed to measure the parameters of the satellite rotation.

Reading from only 1 sensor – magnetometer is required, which is available all the time. Only 2 parameters are required to calculate, α and θ, which could be obtained by a few simple equations. That is a very low computational load and feasible to operate even on the weak onboard CPU.

Stage 3 – Control System Development

Magnetic-torque Actuation Philosophy

Actuating the coils to control the orientation of the satellite seems complicated, but it can be broken down to simply philosophy and understand.

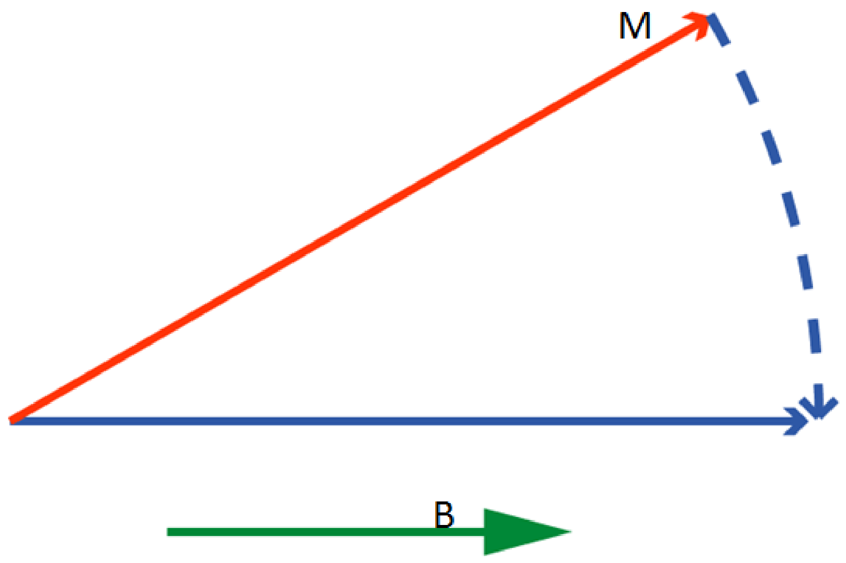


Figure 20 – Actuation Philosophy

For the example that shown in the graph above, the earth magnetic field points towards right; in instance 1, the front of the satellite also points towards the same direction, which is represent by the solid blue line. Sometime later, due to rotation, however, the front of the satellite is no longer pointing to the same direction as the earth's field, and drifts to the red line position. However, if a magnetic moment is generated, with the same direction as the front of the satellite; then the resulting torque will try to reverse the rotation and re-align the front of the satellite to the earth's field.

Therefore, in this method, it is not even required to know direction of the rotation – just take a reading from the magnetometer and generate a magnetic moment in exactly the same direction as where the earth's field is recorded respect to the satellite itself. By doing so, the rotational motion can be slow down regardless of its direction, since there is always a generated torque trying to reverse the satellite's rotation. The best example to visualize is a DC motor without a commuter – when it is powered, the coils will be re-aligned to the same direction as the field.

On the other hand, if a magnetic moment is generated when the earth's field is not perpendicular to the satellite's rotational axis, as shown in the image below; the generated moment would have little effect on slowing the rotation down, but to create another side effect – adding more speed component on other axes by using the limited power from the battery. This is very bad and has to be avoided from the control system.

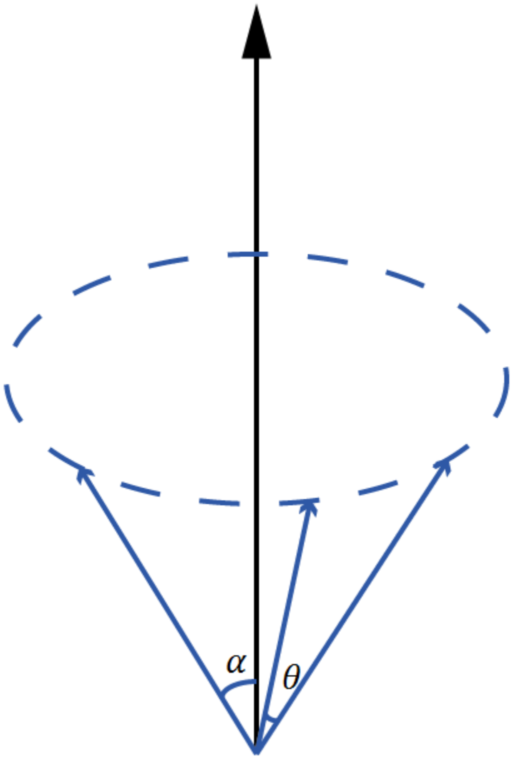


Figure 21 – Cone Shape Rotation

Therefore, the coils should be actuated only when the inclination angle, α, is in the range of 80 to 90 degree – in order to maximize the effect of slowing the rotation down and minimize the side effect that might be created.

Full Control-Action Steps

The full control action cycle is a 6-point cycle.

The first 3 points will be classed as measurement stage – readings from the magnetometer are recorded with a “fix” time-step. With equations that introduced before, θ (speed) and α (inclination) are calculated.

If α is between 80 to 90 degree, then the satellite would enter the next 3 points of the control cycle, which is called the actuation stage. A magnetic moment is generated in the next time-steps, with exactly the same direction as where the earth’s field is recorded at point 2, respect to the satellite itself. By doing so, the generated moment would be approximately perpendicular to the earth’s field and both of them will be perpendicular to the rotational axis, so that the rotational speed could be slowed down and the magnetic-torque efficiency could be maximized.

On the other hand, if α value is not between 80 and 90 degree, then the satellite would skip the following 3 actuating points and start the 6-point control action cycle over again – it will keep waiting in the first 3 points in the control cycle, until the α is within optimal range, then actuate the coils.

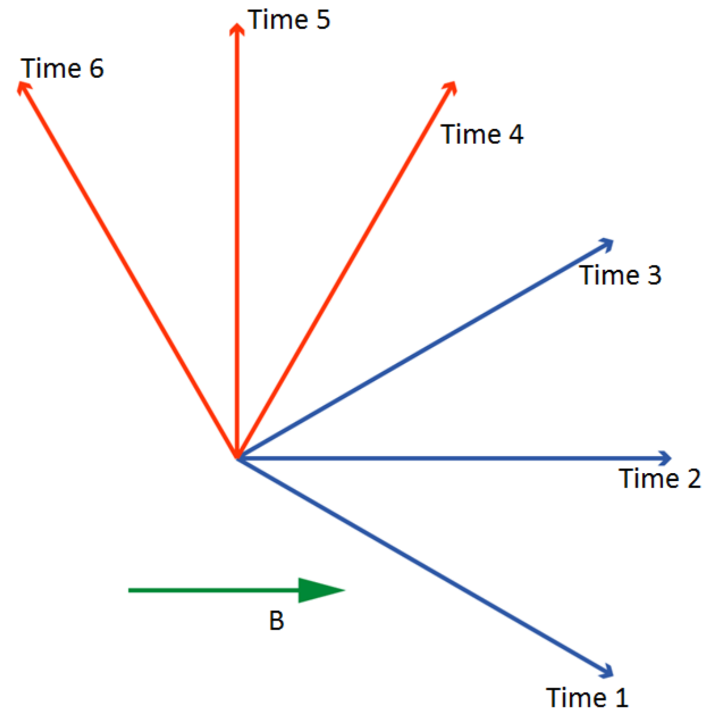


Figure 22 – 6 Points Control System

In addition, anytime the process finishes the first 3 point in the control cycle, the time step will be updated by the latest rotational speed, in order to keep step angle approximately 30 degree which helps with the magnetic-torque efficiency.

Summary

In summary, a de-tumble control system concept has been fully developed. It is a very simple and sophisticated solution for sensing the rotation and actuating the coils – minimal amount of sensor is required, extremely low computational load and do not even need to know the direction of the rotation.

Stage 4 – Simulation

Simulation models are constructed in MATLAB to test the fully developed de-tumble control system, with as close to real physical scenario as possible.

To simulate the whole system, there are three essential parts:

a) emulated environment
b) controller model
c) motion calculator

Environment Emulation

Emulating the environment means reproducing the actual space environment with programs.

Firstly, the orbits for KiwiSAT are simulated by using the orbit's parameters from another satellite that has almost identical orbit. (Vallado & Crawford, 2006) Secondly, based on the positions on the simulated orbits and the earth magnetic field model, the exact earth's fields on each point of the orbit are calculated. (Maus, et al., 2010) Therefore the configurations of the earth's field are obtained.

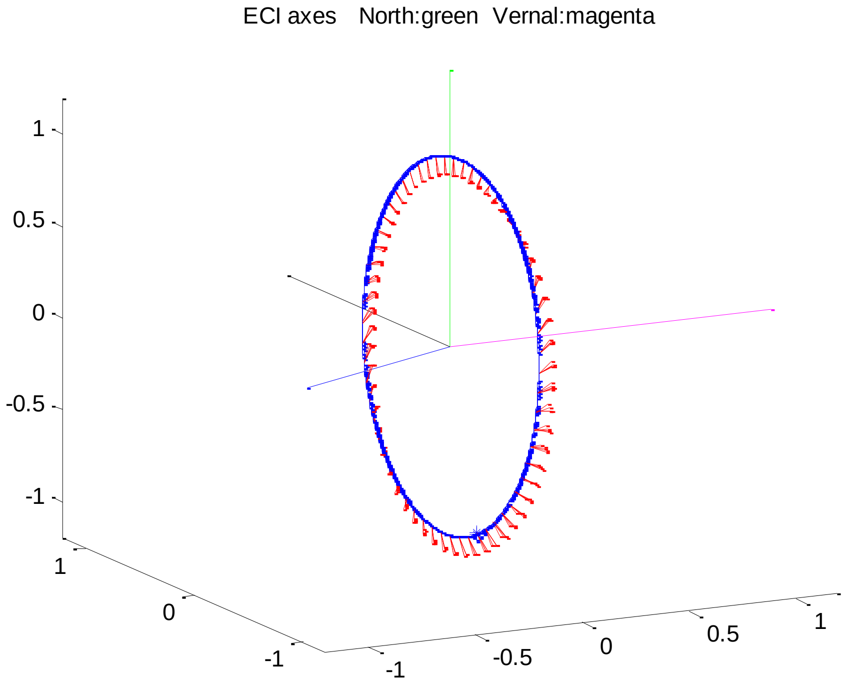


Figure 23 – Simulated Orbits and Earth's Fields

Motion Calculator

Motion calculator, is an engine of implementing Euler’s equations of dynamics motion, which had been covered in the Literature Review. It calculates the attitude of the satellite in the next moment, based on the current attitude and applied torque, which is the resulting product of the generated moment and earth’s field interaction.

Moreover, this is also where the moments of inertia of the satellite are required.

Controller model

The 6-point control cycle system is turned into MATLAB code, which “reads” the magnetometer; preforms calculations for required parameters and “actuates” coil accordingly.

Arming with these 3 parts, proper simulations could then be performed.

Stage 5 – Simulation Results Analysis

Results from the simulation suggest that the de-tumble system is functioning properly – it is able to slow the satellite's rotation of 0.98 rad/sec, which is about 10 RPM, to very low, 0.1 RPM. Its performance is very encouraging – achieving more than the specification required.

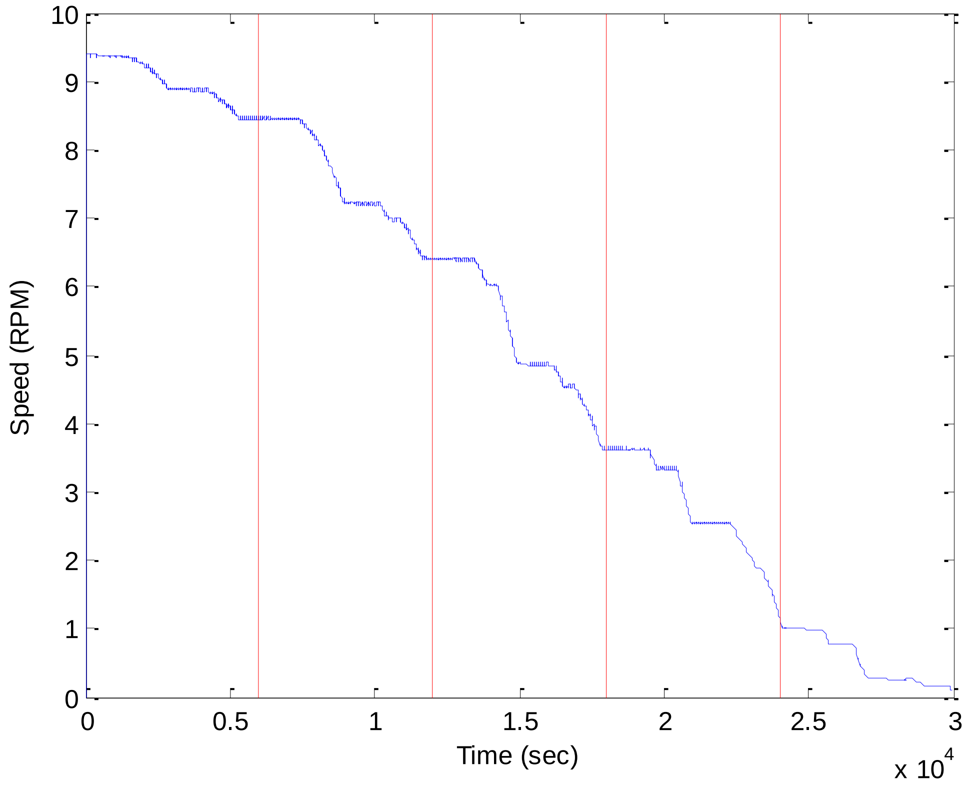


Figure 24 – Simulated Controller Performance

Discussion

Each red line represents the end of orbit; in another word, the de-tumble controller is able to halve the rotation speed within only 3-orbit time, which is a lot better than expected.

In addition, it is noticed that the speed reduces in a shape that looks like stair case. This is due to the nature of the de-tumble system – it waits until an optimal inclination angle, α, is reached to actuate the coils. The speed stays flat while the control system is waiting; the speed reduces when the control system finds opportunities and actuates the coils. Furthermore, it is also discovered that, there are always 4 points/orbit that the earth's field is perpendicular to the rotational axis, which are considered to be opportunities and the coils could actuate by.

Stage 6 – Translate Simulated Control System to C

The last step of the control system development is to convert the simulated control system into C, which will be uploaded to the actual satellite's CPU.

The C program controller has been implemented with sub-routines that are linked to controlling physical hardware, such as getting reading from the magnetometer and actuating the coils with a desired sequence and time step.

Lastly, the C program had been checked and modified by other members in the KiwiSAT Project Team and physically uploaded to the satellite and tested.

Conclusion

In conclusion, a de-tumble controller has been fully developed by following the project stages as mentioned in the introduction section. It has impressive performance and is able to slow the satellite from a high rotation speed of 10 RPM to very low, which satisfied the specification. Lastly, it has been fully verify, tested and ready to be uploaded to the actual satellite for the space mission.

Reference

AMSAT-ZL. (2006). *KiwiSAT Sponsors*. Retrieved 10 2013, from KiwiSAT: http://www.kiwisat.org.nz/sponsors.html

AMSAT-ZL. (2013). *Attitude Determination and Control System*. Retrieved 10 2013, from KiwiSAT: http://www.kiwisat.org.nz/science_1.html

AMSAT-ZL. (2013). *KiwiSAT ADAC*. Retrieved 10 2013, from KiwiSAT: http://www.kiwisat.org.nz/ADAC_Students.html

AMSAT-ZL. (2013, 8). *KiwiSAT Current Status*. Retrieved 10 2013, from KiwiSAT : http://www.kiwisat.org.nz/status.html

AMSAT-ZL. (2013). *The KiwiSAT "Science Package"*. Retrieved 10 2013, from KiwiSAT : http://www.kiwisat.org.nz/science.html

Kennedy, F. (2011). *KiwiSAT Science Package Proposals.*

Kosmotras . (2001). *User's guide for Russia's Dnepr launch vehicle.*

Kosmotras. (n.d.). *Dnepr LV*. Retrieved 10 2013, from Kosmotras: http://www.kosmotras.ru/en/rn_dnepr/

Maus, S., Macmillan, S., McLean, S., Hamilton, B., Nair, M., Thomson, A., et al. (2010). *The US/UK World Magnetic Model for 2010-2015.* New York/Edinburgh: NOAA Technical Report NESDIS/NGDC.

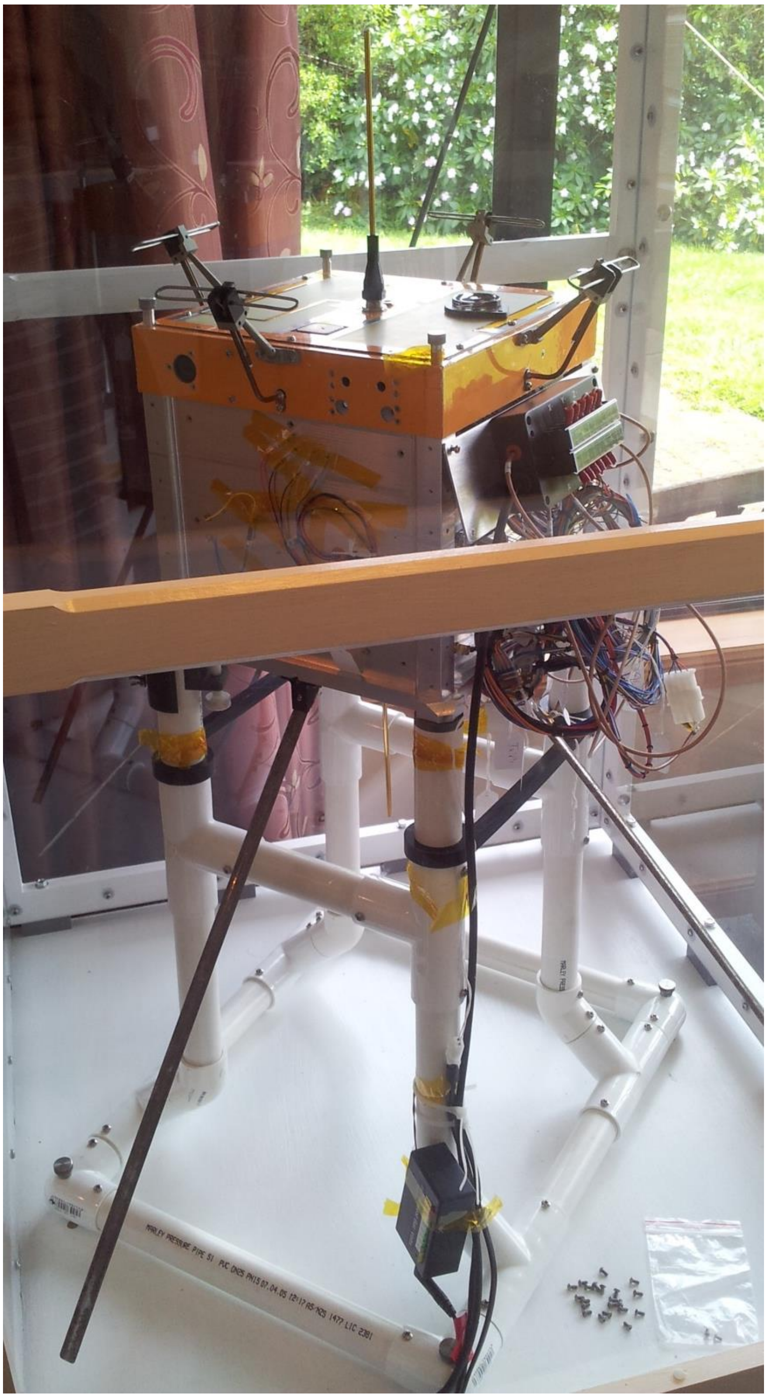
Tate, V. H. (1978). Introduction to Attitude Control. In J. R. Wertz, *Spacecraft Attitude Determination and Control* (pp. 502-509). Dordrecht: D. Reidel Publishing Company.

Vallado, D. A., & Crawford, P. (2006). *SGP4 Orbit Determination.* American Institute of Aeronautics and Astronautics.

Wertz, J. R. (1978). *Spacecraft Attitude Determination and Control.* Dordrecht: D. Reidel Publishing Company.

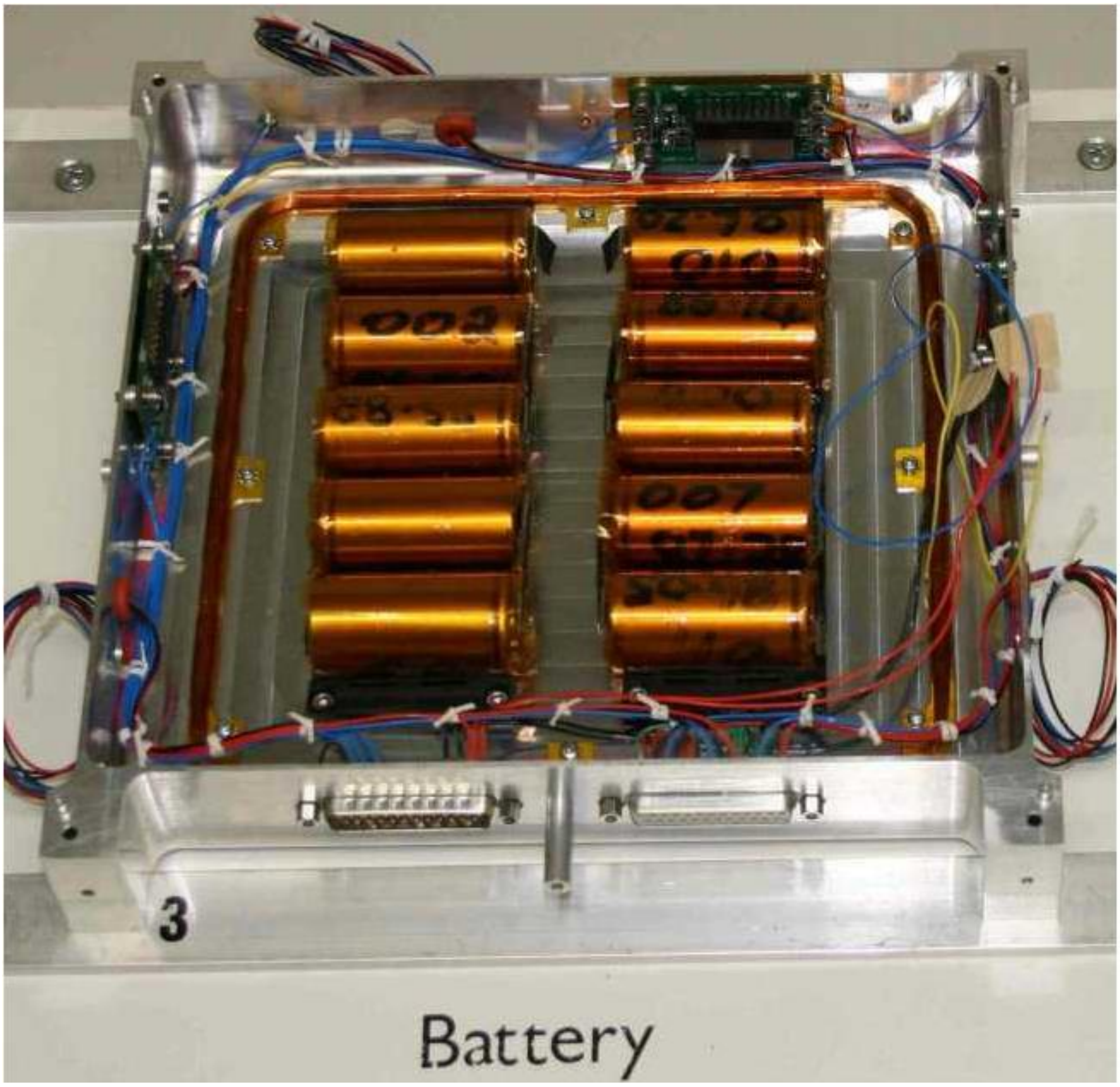
Appendix 1 – Images

The Satellite



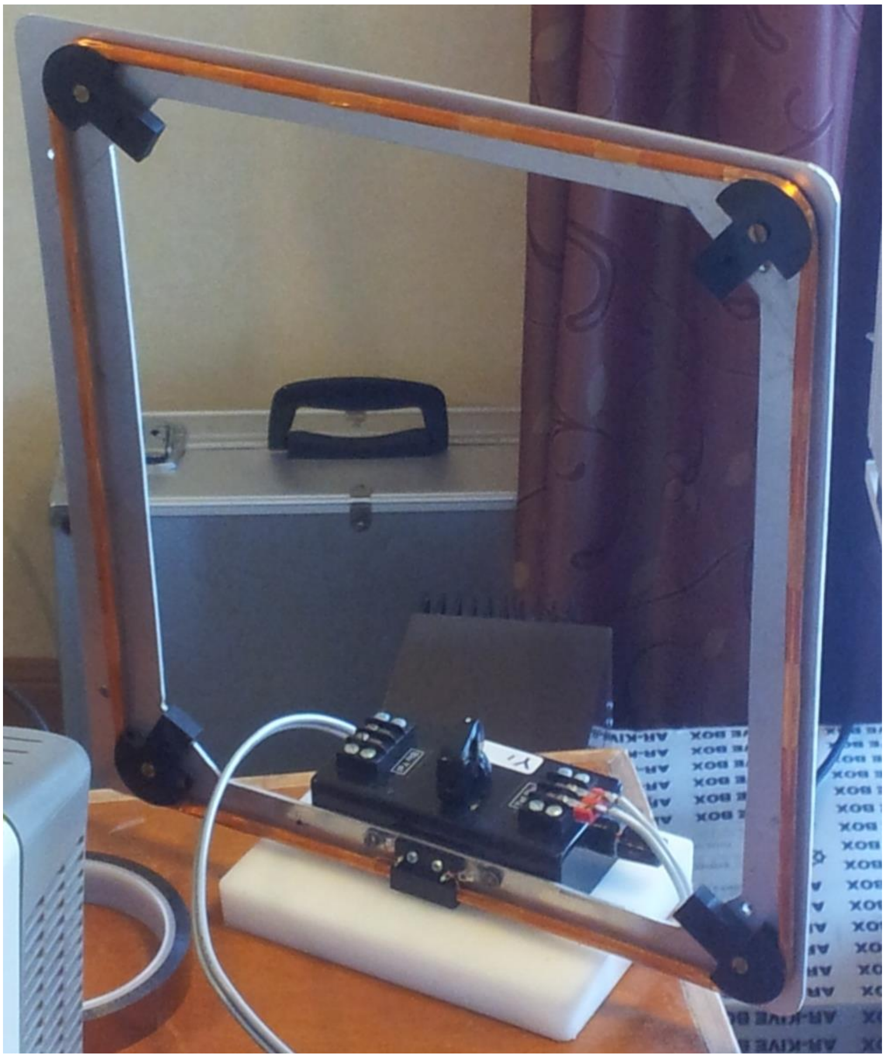
Real Satellite

A part of the KiwiSAT Project Team with the real satellite



Battery Tray

Magnetic Coils



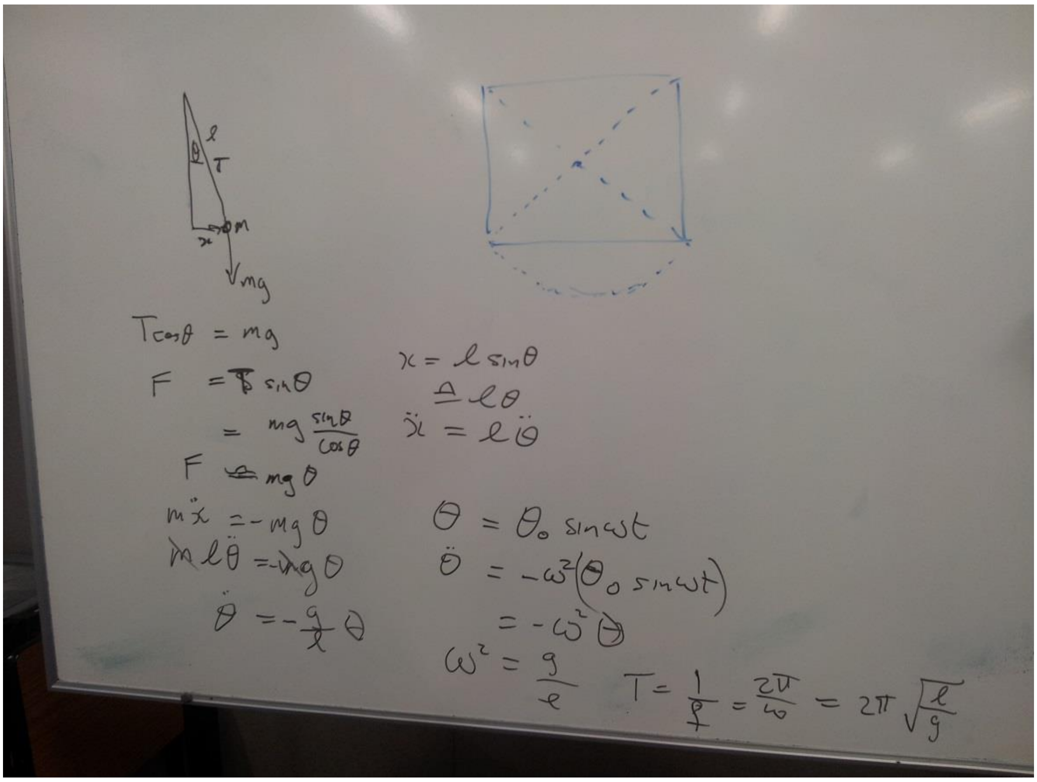
Actual Magnetic Coils: 175x180mm, 110 turns, 26.7Ω, 12V applied

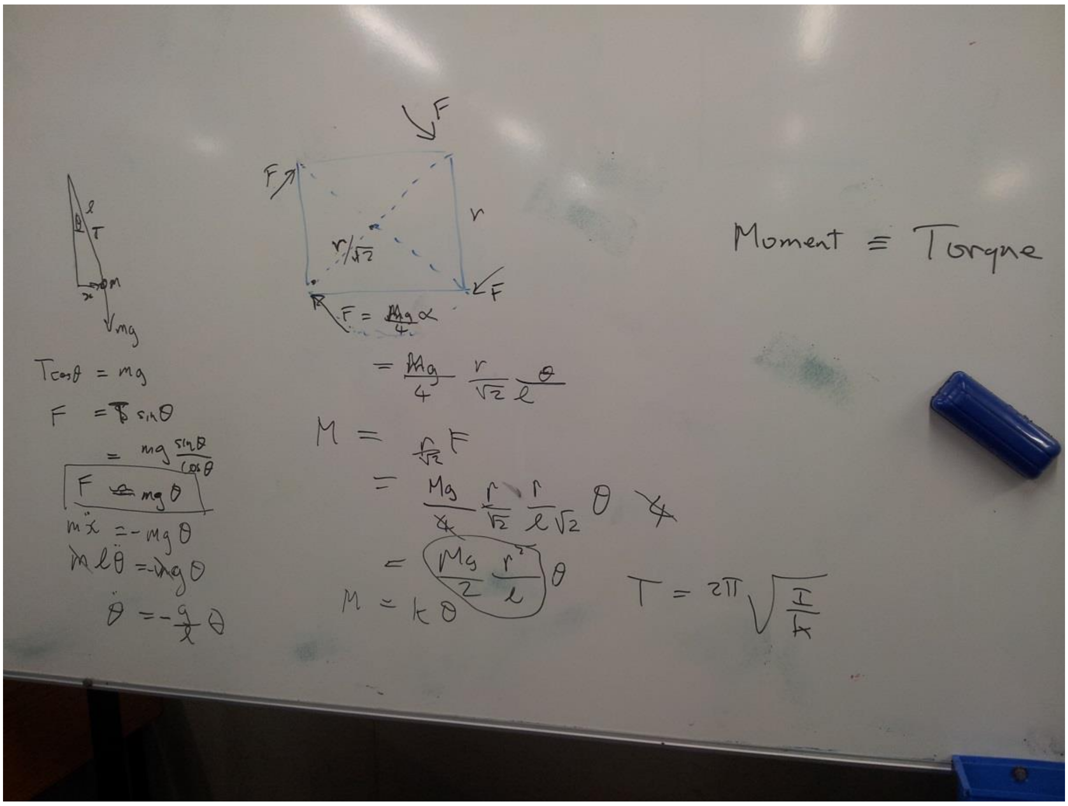
$$I = \frac{V}{R} = \frac{12}{26.75} = 0.45A$$

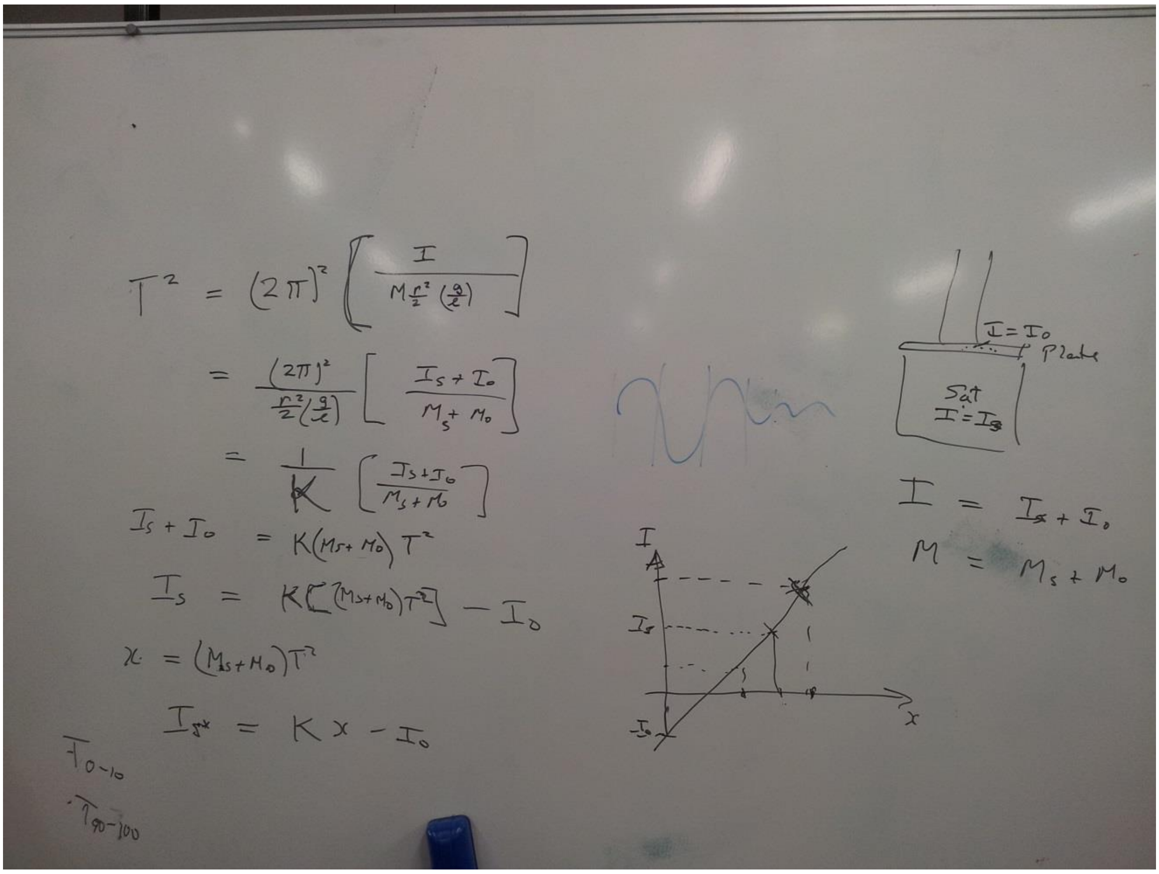
$$A = 0.175 \times 0.18 = 0.0315m^2$$

$$m = NIA = 110 \times 0.45 \times 0.0315 = 1.554Am^2$$

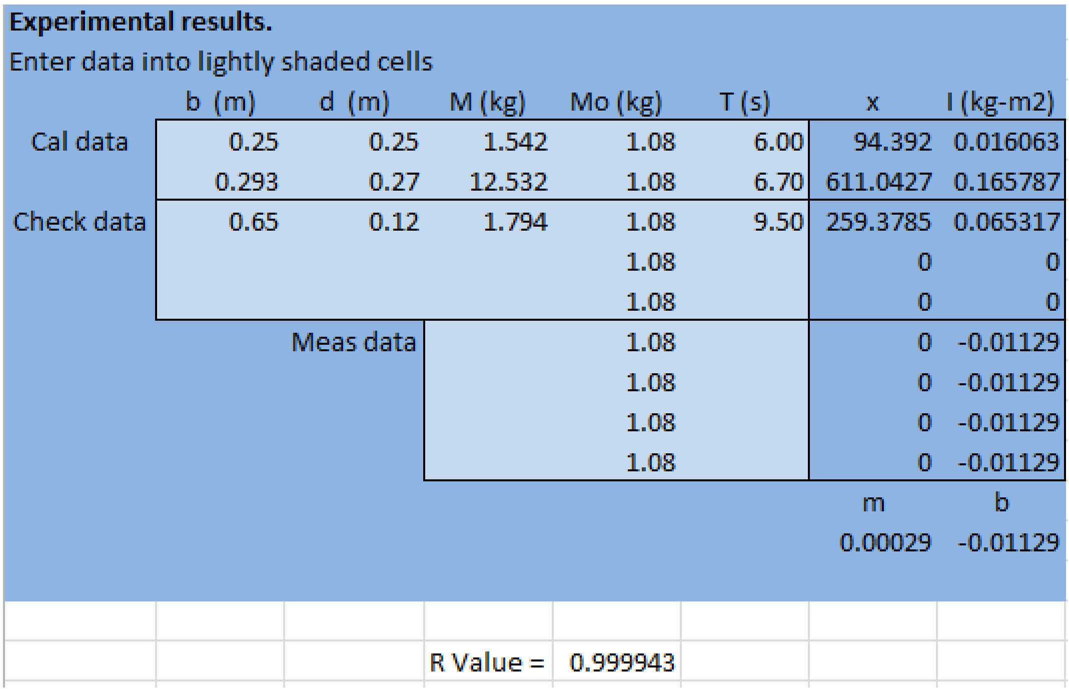
Development of moments of inertia testing method


θ
l
T
m
x
mg
$T\cos\theta = mg$
$F = T\sin\theta$
$= mg\frac{\sin\theta}{\cos\theta}$
$F \triangleq mg\theta$
$m\ddot{x} = -mg\theta$
$\cancel{m}l\ddot{\theta} = -\cancel{m}g\theta$
$\ddot{\theta} = -\frac{g}{l}\theta$
$x = l\sin\theta$
$\triangleq l\theta$
$\ddot{x} = l\ddot{\theta}$
$\theta = \theta_0 \sin\omega t$
$\ddot{\theta} = -\omega^2(\theta_0 \sin\omega t)$
$= -\omega^2\theta$
$\omega^2 = \frac{g}{l}$
$T = \frac{1}{f} = \frac{2\pi}{\omega} = 2\pi\sqrt{\frac{l}{g}}$


θ
l
T
m
x
mg
$T\cos\theta = mg$
$F = T\sin\theta$
$= mg\frac{\sin\theta}{\cos\theta}$
$\boxed{F \triangleq mg\theta}$
$m\ddot{x} = -mg\theta$
$\cancel{m}l\ddot{\theta} = -\cancel{m}g\theta$
$\ddot{\theta} = -\frac{g}{l}\theta$
F
F
F
r
$r/\sqrt{2}$
$F = \frac{Mg}{4}\alpha$
$= \frac{Mg}{4}\frac{r}{\sqrt{2}}\frac{\theta}{l}$
$M = \frac{r}{\sqrt{2}}F$
$= \frac{Mg}{\cancel{4}}\frac{r}{\sqrt{2}}\frac{r}{l\sqrt{2}}\theta\ \cancel{4}$
$= \frac{Mg}{2}\frac{r^2}{l}\theta$
$M = k\theta$
$T = 2\pi\sqrt{\frac{I}{k}}$
Moment ≡ Torque


$T^2 = (2\pi)^2 \left[\frac{I}{M\frac{r^2}{2}\left(\frac{g}{l}\right)} \right]$
$= \frac{(2\pi)^2}{\frac{r^2}{2}\left(\frac{g}{l}\right)} \left[\frac{I_s + I_o}{M_s + M_o} \right]$
$= \frac{1}{K} \left[\frac{I_s + I_o}{M_s + M_o} \right]$
$I_s + I_o = K(M_s + M_o)T^2$
$I_s = K[(M_s+M_o)T^2] - I_o$
$x = (M_s + M_o)T^2$
$I_s = Kx - I_o$
T_{0-10}
T_{90-100}
I
I_s
I_o
x
$I = I_o$ Plate
Sat
$I = I_s$
$I = I_s + I_o$
$M = M_s + M_o$

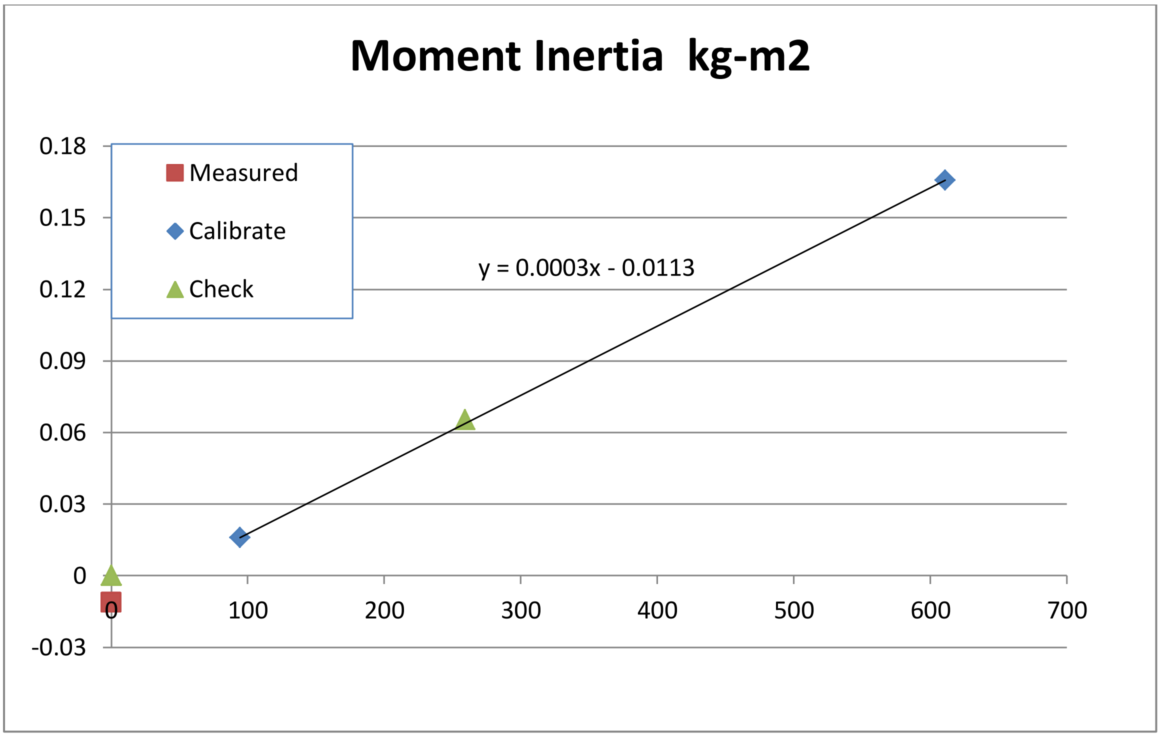
Experiment results:



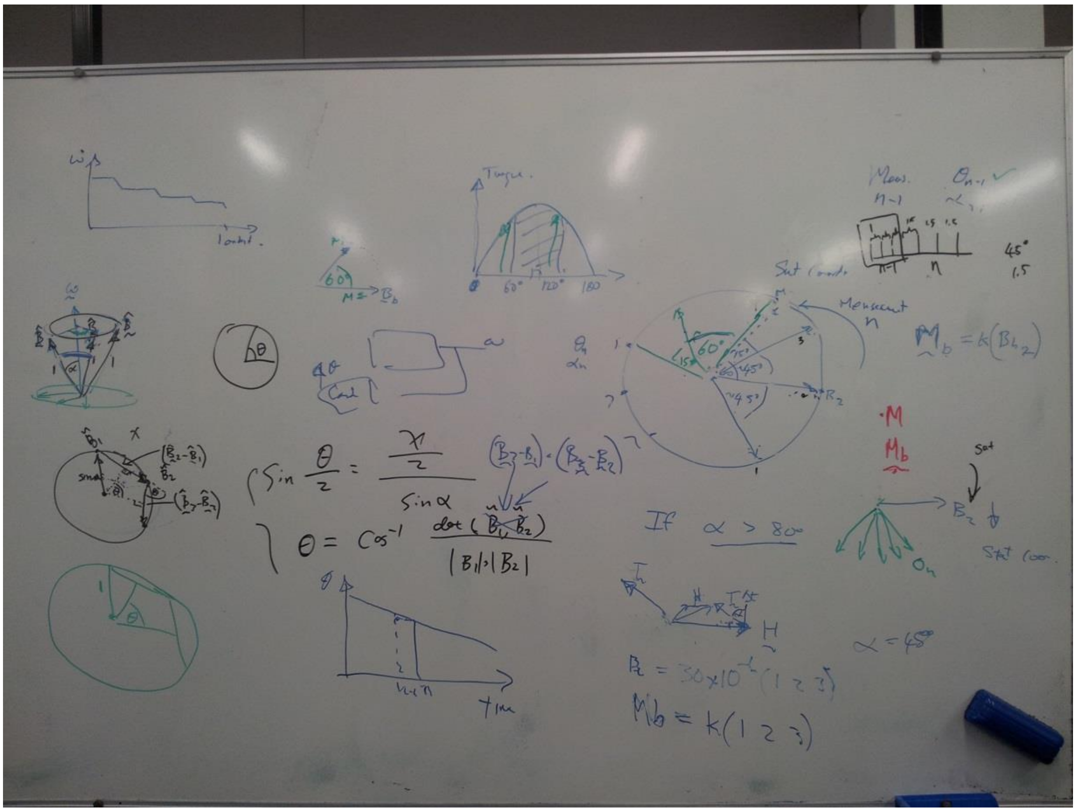
Experimental results.

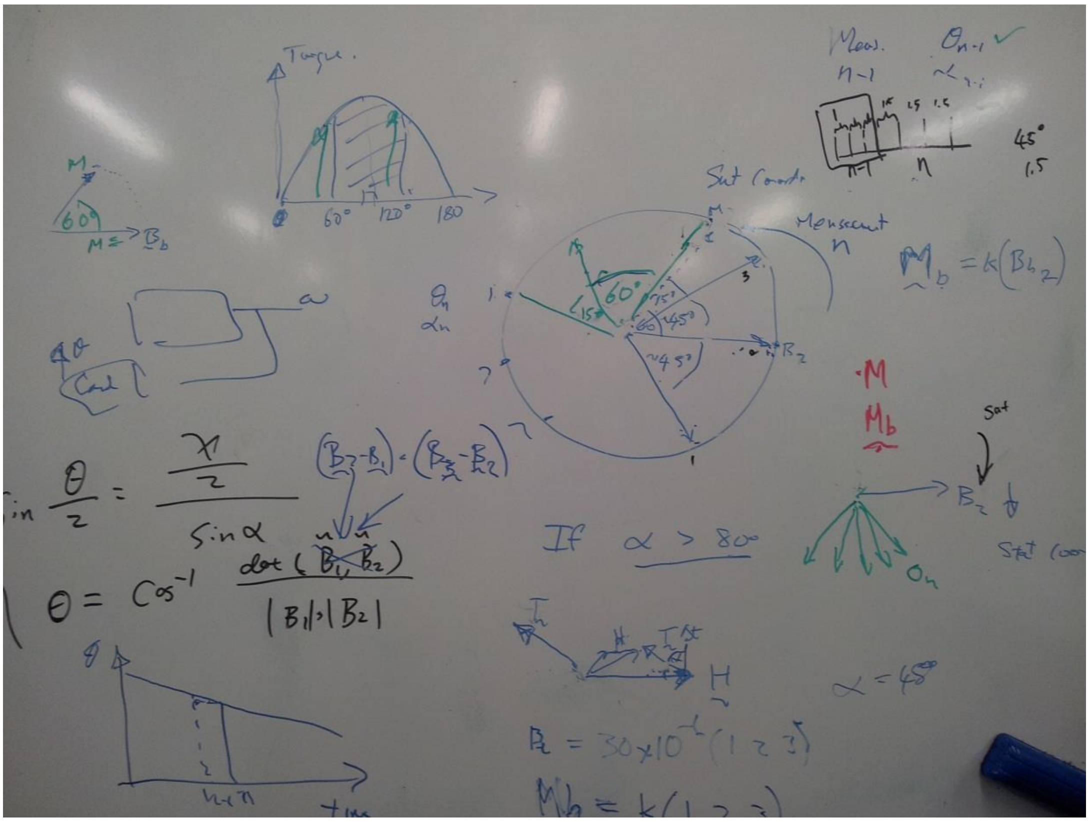
Enter data into lightly shaded cells

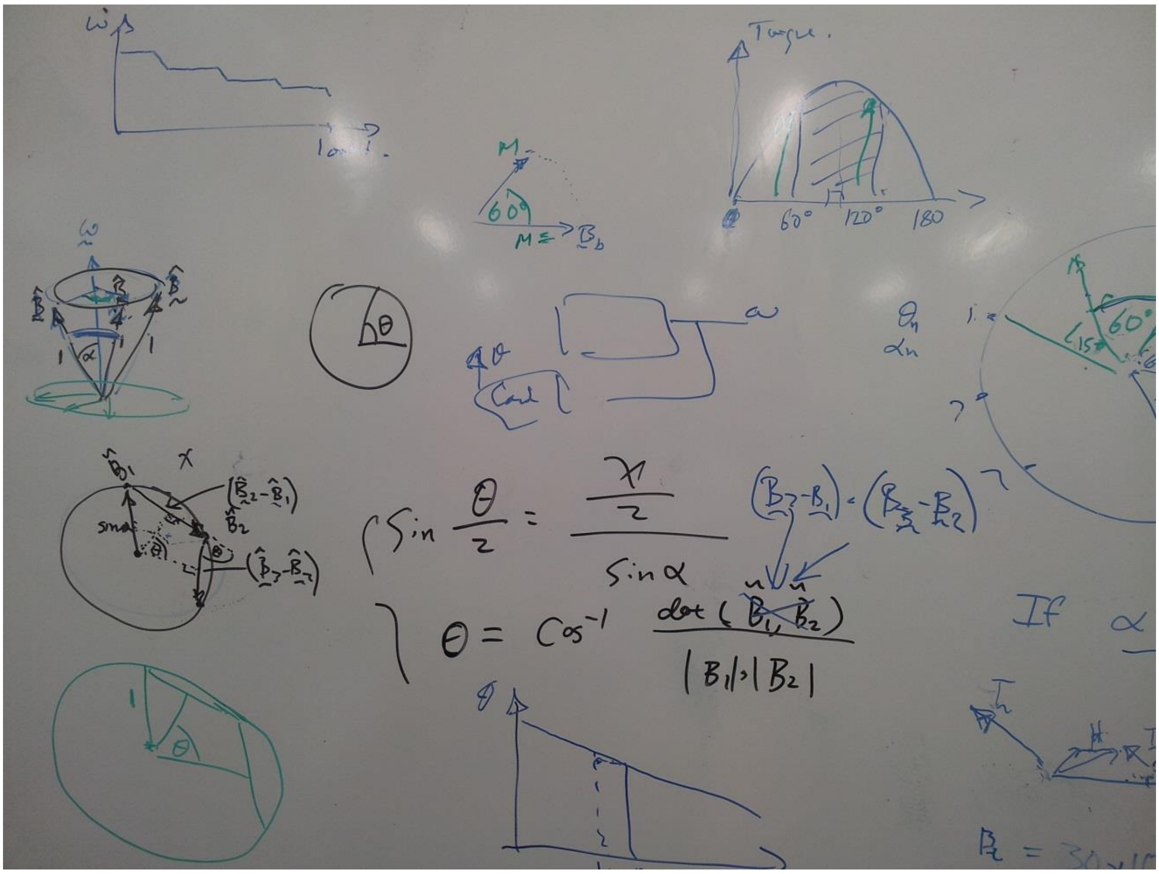
	b (m)	d (m)	M (kg)	Mo (kg)	T (s)	x	I (kg-m2)
Cal data	0.25	0.25	1.542	1.08	6.00	94.392	0.016063
	0.293	0.27	12.532	1.08	6.70	611.0427	0.165787
Check data	0.65	0.12	1.794	1.08	9.50	259.3785	0.065317
				1.08		0	0
				1.08		0	0
		Meas data		1.08		0	-0.01129
				1.08		0	-0.01129
				1.08		0	-0.01129
				1.08		0	-0.01129
						m	b
						0.00029	-0.01129
			R Value =	0.999943			



Rotational measurement and control system development


Torque
60°
120°
180
Meas.
Sat Coord.
Measurement
$M_b = k(B_{L_2})$
M
M_b
$\sin\frac{\theta}{2} = \frac{x/2}{\sin\alpha}$
$\theta = \cos^{-1}\frac{dot(B_1, B_2)}{|B_1||B_2|}$
$(B_2-B_1)\cdot(B_3-B_2)$
If $\alpha > 80$
$\alpha = 45°$
$M_b = k(1\ 2\ 3)$
time


Torque.
0
60°
120°
180
M
60°
M = B_b
ω
θ
Cont
θ_n
α_n
Sat Coords
Measurement
n
Meas.
n-1
θ_n-1 ✓
α_n-1
45°
1.5
M_b = k(B_b2)
M
M_b
Sat
B_2
Sat Coords
θ_n
60°
15
45°
~45°
$\sin\frac{\theta}{2} = \frac{x/2}{\sin\alpha}$
$(B_2 - B_1)\cdot(B_2 - B_2)$
$\theta = \cos^{-1}\frac{\mathrm{dot}(B_1, B_2)}{|B_1|\cdot|B_2|}$
If α > 80
I_n
H
α = 45°
B_2 = 30×10^-1 (1 2 3)
M_b = k(1 2 3)
time


ω
time?
Torque.
0
60°
120°
180
M
60°
M = B_b
ω
B̂
B̂
α
1
1
θ
ω
θ
Cont
θ_n
α_n
B̂1
x
(B̂2 - B̂1)
B̂2
sinα
θ
(B̂2 - B̂2)
1
θ
$\sin\frac{\theta}{2} = \frac{x/2}{\sin\alpha}$
$(B_2 - B_1)\cdot(B_2 - B_2)$
$\theta = \cos^{-1}\frac{\mathrm{dot}(B_1, B_2)}{|B_1|\cdot|B_2|}$
If α
I_n
B_2 = 30×10

Appendix 2 – Simulation Software Code

Field_orbit3000.m

```
%  Programme to generate file of Satellite Position and Earth Mag Field
%   wrt Earth centred celestial reference frame. (Inertial frame).
%   Position in: Rad dist(km) Latc(deg) Lonc(deg)
%   Earth's Mag Field in Bxc, Byc, Bzc, (uT)
%   Calls SGP_AO51,  B_Earth

clear all
%% Ask the user to input the exact date and time of the start of the orbit
fprintf('\nplease enter a date (DAY/MONTH/YEAR) of the start of orbits to
be calculated.\n')
fprintf('Example: 2nd of October, 2012 should be entered as
02/10/2012.\n\n')
da=input('date :','s');
fprintf('\n%s\n',da)
Date = sscanf(da,['%d/']);
%Date(1,1) is day
%Date(2,1) is month
%Date(3,1) is year
n1=sprintf('%d/%d/%d',Date(2,1),Date(1,1),Date(3,1));
n2=sprintf('12/31/%d',Date(3,1)-1);%comepare with a reffernece date - the
last day of the last year
NumDays = daysact(n2,n1); % to calculate "D22-38717" part in the excel

fprintf('\nplease enter a time (HOUR:MIN:SECOND) of the start of orbits to
be calculated.\n')
fprintf('Example: Nine, Forty-Five PM, 40 second should be entered as
21:45:40.\n\n')
ti=input('time :','s');
fprintf('\n%s\n',ti)
Time = sscanf(ti,['%d:']);
%Time(1,1) is hour
%Time(2,1) is min
%Time(3,1) is second
eq_min= (Time(3,1)/60);
min = Time(2,1)+eq_min;
eq_hour = min/60;
hour = Time(1,1) + eq_hour;
eq_day = hour/24;

NASA = NumDays + ((Date(3,1)-2000)*1000)+eq_day
% making a string that contains the full info of the date of the start of
the prbit, in year/month/day/hour/min/second
FullDate =
sprintf('%d/%d/%d/%d/%d/%d',Date(3,1),Date(2,1),Date(1,1),Time(1,1),Time(2,
1),Time(3,1));
% JD = JDtry(FullDate);

%% Read the data file for epoch
filename = '10,1,2013AO_51.txt';
delimiterIn = ' ';
headerlinesIn = 1;
A = importdata(filename,delimiterIn,headerlinesIn);

epoch = sscanf( char(A.textdata(4)),'%f32');% Time after epoch
(13/9/2012),time of satellite's data taken
 day = NASA - epoch;
```

```
fprintf('That is %d days after the epoch,',day)
fprintf('Example, 1.986862e+001 means 19 days with a few more hours.\n\n\n')

if(day>15)
    run=0;
    fprintf('The Keps paramaters are a bit old, newer data can be download
from: \n')
    fprintf('http://www.amsat.org/amsat/ftp/keps/current/nasa.all')
    fprintf('\nhowever, would you still like to continue the
calculation?\n')
    temp=input('Y/N :','s');
    temp2 = sscanf(temp,['%c']);
    if(temp2=='Y' || temp2=='y')
        run=1;
    else
        run=0;
    end
elseif(day>0)
    run=1;
else
    fprintf('The time of the start of the orbit is before the Keps
paramaters measured, older data can be download from: \n')
    fprintf('http://www.amsat.org/amsat/ftp/keps/current/nasa.all')
    fprintf('\nhowever, would you still like to continue the
calculation?\n')
    temp=input('Y/N :','s');
    temp2 = sscanf(temp,['%c']);
    if(temp2=='Y' || temp2=='y')
        run=1;
    else
        run=0;
    end
end

%% Execute calculation
if(run==1)

    fprintf('Busy calculating.....\n\n\n')

    % Enter the orbit points required
    %day = 10; % 10 days after epoch (23 May 2006)
    % Same orbit as in Predict_b2
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %change over to 10 a default
    dt = 10;  % one orbits of AO51
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %% Call the function to get the position of the satelite,by given time
    Pos = SGP_AO51_read_text_JDcal(day,600*5,dt,FullDate);
    %Pos's data is in cerlestial or Earth-fix longtitude and latitude
system
    Fld = zeros(600*5,3);
    BEarth = zeros(600*5,3);

    % calculate field over orbit
    for n = 1:600*5
        %% Find field
        % returns the Earth magnetic field in the Earth-fix spherical
axis(r, theta and phi)
        BEarth(n,1:3) = B_Earth2010(Pos(n,1),90-Pos(n,2),180+Pos(n,4));
        theta = (pi/180)*(90-Pos(n,2));
        phi = (pi/180)*Pos(n,3);
        H = [sin(theta)*cos(phi) cos(theta)*cos(phi) -sin(phi);...
            sin(theta)*sin(phi) cos(theta)*sin(phi) cos(phi);...
            cos(theta) -sin(theta) 0];
        Fld(n,1:3) = (H*Bearth(n,1:3)')';

    end
    %% Save the data to temp file "Pos_Fld"
    Pos_Fld = [Pos(:,1:3),Fld,Pos(:,5)];%radial dist(km), latc (deg), longc
(deg), 3 axis magnetic field in cerlestial coordinate and JD

    save temp1 Pos_Fld

    %% Generating the SatPosition file for Tim
    [nrows,ncols]=size(Pos_Fld);
    filename = 'SatPosition.txt';
    fid = fopen(filename,'W');
    time = 0;
    for row = 1:nrows
        fprintf(fid,
'%d\t%f\t%f\t%f\r\n',time,Pos(row,1),Pos(row,2),Pos(row,3));
        time = time + dt;
    end
    fclose(fid);

    fprintf('Calculations have been completed,\n')
    fprintf('For display the Orbits and Magnetic Field results, please run
Examine3000.m\n')
    fprintf('For conducting the Simulations of when magnetic coils are
energised, please run Kiwisatcont4b_sim.m\n')
else
    fprintf('Programme ends.\n')
end
```

10,1,2013AO_51.txt

1 28375U 04025K   13003.89155755 .00000122 00000-0 49230-4 0 4541

2 28375 098.1762 326.1541 0083310 152.5761 207.9855 14.410513144 47672

SGP_AO51_read_text_JDcal.m

```
function Pos = SGP_AO51_read_text_JDcal(day,n,dt,FullDate)
% Function to give radial dist(km), latc (deg), longc (deg), lon (deg)
% at points along satellite orbit.
%    Start time after epoch : day  (unit = day)
%     Number of points : n
%     Time increment between points = dt (unit = sec)
%     Uses SGP model
%    Calls LoncGrch

%% Inport Orbit's Data from the data file
filename = '10,1,2013AO_51.txt';
delimiterIn = ' ';
headerlinesIn = 1;
A = importdata(filename,delimiterIn,headerlinesIn);
```

```
%   Inclination = A.data(1, 3)
%   RAAN=A.data(1, 4)
%   Eccentricity =A.data(1, 5)/10000000
%   Argument_of_Perigee=A.data(1, 6)
%   Mean_Anomaly=A.data(1, 7)
%   Mean_Motion=A.data(1, 8)
%   Revolution_Number_at_epoch =A.data(1, 9)

%  Rate_of_change_of_Mean_motion = sscanf( char(A.textdata(5)),'%f')
%  epoch = sscanf( char(A.textdata(4)),'%f32')

% Orbit parameters (KEPs) at epoch
to = sscanf( char(A.textdata(4)),'%f32');   % Time after epoch
(13/9/2012),time of satellite's data taken

eo = A.data(1, 5)/10000000;     % Eccentricity of orbit
io = A.data(1, 3);          % Inclination of orbit (deg)
Mo = A.data(1, 7);          % Mean Anomaly (deg)
no = A.data(1, 8);      % Mean motion (rev/day)

dnodt = sscanf( char(A.textdata(5)),'%f');  % Rate of change of Mean motion
(rev/day/day)
d2nodt2 = 0;            % Acceleration of Mean motion (rev/day/day/day)
wo = A.data(1, 6);          % ARGument of Perigee (deg)
Wo = A.data(1, 4);          % RAAN (deg)

% Getting the full info of the date of the start of the prbit, in
year/month/day/hour/min/second
DateJD = sscanf(FullDate,['%d/']);
%DateJD(1,1) is year
%DateJD(2,1) is month
%DateJD(3,1) is day
%DateJD(4,1) is hour
%DateJD(5,1) is min
%DateJD(6,1) is second
JD =
juliandate([DateJD(1,1),DateJD(2,1),DateJD(3,1),DateJD(4,1),DateJD(5,1),Dat
eJD(6,1)]);
%the difference of the JD between every step size
JDstep = (juliandate([2012,10,10,10,10,10]) -
juliandate([2012,10,10,10,10,9]))*dt;
JDcount = JD;



%% Find celestial longitude of Greenwich at time of epoch
ne = 360.98564736629;   % Rotation of earth (deg/day)
lonc_grn = LoncGrch(to);

%% Constants
ae = 6378.135;          % Earth radius (km)
a1 = (8681660.4/no)^(2/3);      % Approx major semi-axis (km)
cJ2 = 0.5*0.00108263*(ae/a1)^2;    % 0.5*J2*(ae/a1)^2
J3rJ2 = -0.00234507;               % Ratio of J3 to J2
d1 = 1.5*cJ2*(3*cos(pi*io/180)^2-1)/(1-eo^2)^1.5;
ao = a1*(1-d1/3-d1^2-(134/81)*d1^3);    %  Major semi-axis at epoch
po = ao*(1-eo^2);
qo = ao*(1-eo);
Lo = Mo + wo + Wo;
dWdt = -3*cJ2*(a1/po)^2*360*no*cos(pi*io/180);              % deg per day
dwdt = 1.5*cJ2*(a1/po)^2*360*no*(5*cos(pi*io/180)^2-1);   % deg per day
```

```
Pos = zeros(n,5);
for m = 1:n
    t = day + (m-1)*dt/86400;%86400 seconds per day

    %% Secular perturbations
    a = ao*(no/(no+dnodt*t+0.5*d2nodt2*t^2))^(2/3);     % Major sem-axis
    if a > qo, e = 1-qo/a; else e = 1e-6; end
    p = a*(1-e^2);
    Wso = Wo + dWdt*t;                      % RAAN
    wso = wo + dwdt*t   ;                   % Arg perigee
    Ls = Lo + (360*no+dwdt+dWdt)*t + 360*((dnodt/2)*(t^2) +
(d2nodt2/6)*(t^3)); % Full anomaly

    %% Convert angles to radians
    Lsr = (pi/180)*Ls; Wsor = (pi/180)*Wso;
    wsor = (pi/180)*wso; ior = (pi/180)*io;
    ayNSL = e*sin(wsor) - 0.5*(J3rJ2)*(ae/p)*sin(ior);
    axNSL = e*cos(wsor);
    Lr =Lsr -
0.25*(J3rJ2)*(ae/p)*axNSL*sin(ior)*((3+5*cos(ior))/(1+cos(ior)));

    %% Solve Keplers equn
    Ur = Lr - Wsor;         % Eccentric anomaly (initial)
    Ewr = Ur;
    Ewor = Ewr+1;
    while abs(Ewr-Ewor) > 0.0000001
        Ewor = Ewr;
        dEwor = (Ur-ayNSL*cos(Ewor)+axNSL*sin(Ewor)-Ewor)/(1-
ayNSL*sin(Ewor)-axNSL*cos(Ewor));
        Ewr = Ewor + dEwor;
    end

    %% Intermediate partially osculating quantities
    ecosE = axNSL*cos(Ewr) + ayNSL*sin(Ewr);
    esinE = axNSL*sin(Ewr) - ayNSL*cos(Ewr);
    eL2 = axNSL^2 + ayNSL^2;
    pL = a*(1-eL2);
    r = a*(1-ecosE);
    sinu = (a/r)*(sin(Ewr)-ayNSL-axNSL*esinE/(1+sqrt(1-eL2)));
    cosu = (a/r)*(cos(Ewr)-axNSL+ayNSL*esinE/(1+sqrt(1-eL2)));
    ur = atan(sinu/cosu);
    if cosu<0 ur = ur + pi; end

    %% short period pertubations are now included
    rk = r + 0.5*cJ2*(a1^2/pL)*sin(ior)^2*cos(2*ur);
    ukr = ur - 0.25*cJ2*(a1/pL)^2*(7*(cos(ior))^2-1)*sin(2*ur);
    Wkr = Wsor + 1.5*cJ2*(a1/pL)^2*cos(ior)*sin(2*ur);
    ikr = ior + 1.5*cJ2*(a1/pL)^2*sin(ior)*cos(ior)*cos(2*ur);

    %% Calculate unit postion vector wrt Celestial XYZ axes
    M = [-sin(Wkr)*cos(ikr) cos(Wkr)*cos(ikr) sin(ikr)];
    N = [cos(Wkr) sin(Wkr) 0];
    U = M*sin(ukr) + N*cos(ukr);

    %% Calculate position in terms of Radial dist, latC & onc
    rad = rk;
    latc = (180/pi)*asin(U(3));
    rxy = sqrt(U(1)^2 + U(2)^2);
    lonc = (180/pi)*acos(U(1)/rxy);
    if U(2)< 0
        lonc = -lonc;
    end

    %% Transform Celestial longitude to terrestrial longitude by -
    %     subtracting celestial longitude of Greenwich and angle of
    %     earth's rotation since epoch.
    lon = lonc - lonc_grn - ne*t;
    lon = lon - 360*fix(lon/360);       % Bring into 360 deg range
    if lon <= -180
        lon = lon + 360;    % Bring into -180 > +180 range
    elseif lon > 180
        lon = lon - 360;
    else end

    Pos(m,:) = [rad,latc,lonc,lon,JDcount];
    JDcount = JDcount + JDstep;
end
```

B_earth2010.m

```
function field=B_earth2010(r,theta,phi)
% Function to compute Radial, Theta, Phi components of
%       Earth's Magnetic Field (nT) at single point.
% Note theta = 90-latitude, phi = 180+longitude
%       These  theta & phi are for sph polars and phi is different to one
used in IGRF10.
% Note at Auckland,  latitude = -36.6,  longitude = 175

latrad = pi*(90-theta)/180;
longrad = pi*(phi)/180;    % Note 180deg ADDED

% World Magnetic Model coefficients
t = 2.5;    % Decimal time in yr from Yr 2010, use for magnetic field
calculation
wmm_coef = [-294966 0 116 0;  -15863 49444 165 -259; -23966 0 -121 0;
    30261 -27077 -44 -225; 16686 -5761 19 -118;  13401 0 4 0;
    -23262 -1602 -41 73;   12319 2519 -29 -39;    6340 -5366 -77 -26;
    9126 0 -18 0;    8089 2864 23 11;    1667 -2112 -87 27;
    -3571 1643 46 39;   894 -3091 -21 -8;    -2309 0 -10 0;
    3572 446 6 4;    2003 1889 -18 18;   -1411 -1182 -10 12;
    -1630 0 9 40;    -78 1009 10 -6;   728 0 -2 0;
    686 -208 -2 -2;    760 441 -1 -21;   -1414 615 20 -4;
    -228 -663 -17 -6;   132 31 -3 5;    -779 550 17 9;
    805 0 1 0;    -751 -579 -1 7;    -47 -211 -6 3;
    453 65 13 -1;   139 249 4 -1;    104 70 3 -8;
    17 -277 -7 -3;    49 -33 6 3;   244 0 -1 0;
    81 110 1 -1;    -145 -200 -6 2;   -56 119 2 4;
    -193 -174 -2 4;   115 167 3 1;  109 70 3 -1;
    -141 -108 -6 4;    -37 17 2 3;   54 0 0 0;
    94 -205 -1 0;   34 115 0 -2;   -52 128 3 0;
    31 -72 -4 -1;    -124 -74 -3 1;  -7 80 1 0;
    84 21 -1 -2;   -85 -61 -4 3;   -101 70 -2 2;
    -20 0 0 0;    -63 28 0 1;   9 -1 -1 -1;
    -11 47 2 0;    -2 44 0 -1;    25 -72 -1 -1;
    -3 -10 -2 0;    22 -39 0 -1;   31 -20 -1 -2;
    -10 -20 -2 0;    -28 -83 -2 -1;   30 0 0 0;
    -15 2 0 0;   -21 17 0 1;   17 -6 1 0;
    -5 -18 0 1;    5 9 0 0;   -8 -4 0 1;
    4 -25 0 0;   18 -13 0 -1;   1 -21 0 -1;
    7 -19 -1 0;   38 -18 0 -1;    -22 0 0 0;
    -2 -9 0 0;   3 3 1 0;   10 21 1 0;
    -6 -25 -1 0;    9 5 0 0;    -1 6 0 1;
    5 0 0 0;   -4 1 0 0;    -4 3 0 0;
    2 -9 0 0;   -8 -2 -1 0;   0 9 1 0];
wmm_coef = 0.1*wmm_coef;

latinc = pi/360;    % Increment used for calculating gradient
a = 6371.2;         % Standard Earth Reference radius in km

% Find X, Y, Z components
Xt=0;
Yt=0;
Zt=0;
for n=1:12
    SP=legendre(n,sin(latrad),'sch');
    SPgradu=legendre(n,sin(latrad+latinc),'sch');
    SPgradl=legendre(n,sin(latrad-latinc),'sch');
    nn = n*(n+1)/2;
    Xm=0;
    Ym=0;
    Zm=0;
    for m=0:n
        gcos = (wmm_coef(nn+m,1) + t*wmm_coef(nn+m,3))*cos(m*longrad);
        hsin = (wmm_coef(nn+m,2) + t*wmm_coef(nn+m,4))*sin(m*longrad);
        gsin = (wmm_coef(nn+m,1) + t*wmm_coef(nn+m,3))*sin(m*longrad);
        hcos = (wmm_coef(nn+m,2) + t*wmm_coef(nn+m,4))*cos(m*longrad);
        Xm = Xm + (gcos+hsin)*(((-1)^m)*SPgradu(m+1)-((-
1)^m)*SPgradl(m+1))/(2*latinc);
        Ym = Ym + m*(gsin-hcos)*((-1)^m)*SP(m+1);
        Zm = Zm + (gcos+hsin)*((-1)^m)*SP(m+1);
    end
    Xt=Xt+Xm*(a/r)^(n+2);
    Yt=Yt+Ym*(a/r)^(n+2);
    Zt=Zt+Zm*(n+1)*(a/r)^(n+2);
end
Bradial = Zt;                   % Note + , because outward
Btheta = Xt;                    % Note + , because in theta dirn (ie south)
Bphi = Yt/cos(latrad);      % Note + , because in phi dirn (ie east)
field=[Bradial Btheta Bphi];
```

Examine3000.m

```
% Programme to examine Position & Mag Field of an orbit
%   wrt earth centred inertial reference frame. (Celestial cord).
%   Rad dist(km) Latc(deg) Lonc(deg) Bx(nT) By(nT) Bz(nT)
%   Loads previously generated data file temp1

%% Load the data
clear all
clf
num = 600*5;                % Number of data points in orbit, for 5 times
load temp1 Pos_Fld; % Previosly calculated orbit and field.  See
Field_orbit
B = zeros(num,4);
B(:,1:3) = Pos_Fld(:,4:6);  % Earth's magnetic field (in celestial XYZ axis)
% Calculate magnitude of B at each point.
for n=1:num,
    B(n,4)=norm(B(n,1:3));
end %calculate the magnitude of the magnetic field and store it in the 4
column

%% Plot magnitude of earth's field vs latitude
figure(1)
hold on
plot(Pos_Fld(:,2),B(:,4))%the second column of Pos_Fld is Latitude
plot(Pos_Fld(1,2),B(1,4),'k*')%starting point, the first value in Latitude
and Mag.field
axis([-90 90 0 5E4])
title('Magnitude of Earths Field (nT)  v  Latitude')
% title('')
hold off

%% Plot point to point differences of earth's field
D = zeros(num,4);
for n=2:num
   for nn=1:3
      D(n,nn)=B(n,nn)-B(n-1,nn);%calculated the differences between each
point
   end
   D(n,4) = norm(D(n,1:3));% the norm of the difference, which has already
taken the change of direction into account
end
figure(2)
plot(1:num,D(:,4))
title('Point-to-point differents of Earths Field (nT)  v  10sec')

%% Put orbit into xyz cordinates ?? TickTick
% how to obtain the x y and z coordinate by latc and longc?
% what does it actually mean? convert the celartial system to...?
thetar = (pi/180)*Pos_Fld(:,2);%cerlestial latitude, not bolar angel
phir = (pi/180)*Pos_Fld(:,3);%cerlestial longtitude, not bolar angel
x = cos(thetar).*cos(phir);%where is the radius? should be multiply
together as well.
y = cos(thetar).*sin(phir);%get the satellite's position in cerlestial
corrdinate by using cerlestial latitude and longtitude
z = sin(thetar);
figure(3)
hold on
plot3(x,y,z,'b')
plot3(x(1:2),y(1:2),z(1:2),'b*')
```

```
axis([-1.2 1.2 -1.2 1.2 -1.2 1.2])

%% Plot field vectors at every 10 points. ??
xd = zeros(2,1);
yd=xd;
zd=xd;
for n=1:10:num
   xd(1) = x(n);
   yd(1) = y(n);
   zd(1) = z(n);
   xd(2) = xd(1) + 0.1*B(n,1)/B(n,4);% is is really necessary to be devided
by the norm? i think doesn't matter the direction, but just have got
smaller number easier to display
   yd(2) = yd(1) + 0.1*B(n,2)/B(n,4);
   zd(2) = zd(1) + 0.1*B(n,3)/B(n,4);
   plot3(xd,yd,zd,'r')%3D plotting the vector line from point (xd1,yd1,zd1)
to point (xd2,yd2,zd2)
end
s=1.5;
plot3([0 s]',[0 0]',[0 0]','m')
plot3([0 0]',[0 s]',[0 0]','k')
plot3([0 0]',[0 0]',[0 s]','g')
title('ECI axes   North:green  Vernal:magenta')
% Plot RAAN = 196.8+9.56 = 206.4
plot3([0,cos((pi/180)*206.4)]',[0,sin((pi/180)*206.4)]',[0 0]','b')%what it
is for? how to calculate that? why not just read the RAAN from the doc?
view(-45,20) % 206.4 > -90-63.6  Therefor use (-63.6,0) for side on view
hold off

%% Plot the actual magnetic field magnetude through the whole process
figure(4)
plot(B(:,4))
title('Magnitude of Earths Field (nT)  v  10sec')

%% plot the actual magnetic field magnetude difference vs tiem (or the rate
of change)

D2 = zeros(num,1);
figure(5);
for n = 2:num
   D2(n,1) = B(n,4)-B(n-1,4);
end
plot(D2(:,1))
title('rate of change of the earth Magnitude Field (nT)  v  10sec');

%% plot the actual magnetic field magnetude difference vs latitude (or the
rate of change)
figure(6);
plot(Pos_Fld(:,2),D2(:,1))
title('rate of change of the earth Magnitude Field (nT)  v  Latitude');
```

LaDetum_msPlateForm2.m

```
%% Initialization and Data Loading
clear all
tlim = 29991;
nn = tlim/1;           % Number of points in the orbit
```

```
% Initial attitude and angular velocity of satellite
qo = [0.3711, 0.3711, -0.3711, -0.7660];


qo = qo/norm(qo);
wbo =  1*[0.6 0.5 0.6]';    % Initial angular velocity
TorqK = 80;

% Satellite Moments of Inertia
Inertia = [0.15 0.15 0.15]';    % Pr Mmts of Inertia of satellite around CG
k = zeros(3,1);                 % Normalised inertia coefficients
k(1) = (Inertia(2)-Inertia(3))/Inertia(1);
k(2) = (Inertia(3)-Inertia(1))/Inertia(2);
k(3) = (Inertia(1)-Inertia(2))/Inertia(3);

Gxm = -0.5;  Gym = 0.5;  Gzm = 0.5;%gain and offset of the magnitometer
Vxmo = 2.42;  Vymo = 2.48; Vzmo = 2.53;

% load Earth's Mag Field
load temp1 Pos_Fld; % Previously calculated orbit and field.  See
Field_orbit
%Pos_Fld is in such a formate: radial dist(km), latc (deg), longc (deg),
magnetic field in 3 axis and JD
%% Do you want to play animation?
Display = input('Do you want to play animation? (Y/N )','s');
Display = sscanf(Display,['%c']);
if(Display=='Y' || Display=='y')
    Displayrun=1;
else
    Displayrun=0;
end

%% Getting Earth's magnetic field information
B = zeros(3000,3);
B(:,1:3) = 1e-9*Pos_Fld(1:3000,4:6);
for jj=1:3000
    Bcn(jj,4) = norm(B(jj,:));
    Bcn(jj,1:3) = B(jj,1:3)/Bcn(jj,4);
end

for ii=1:3
    kk=1;
    for jj=1:2999
        step=(B(jj+1,ii)-B(jj,ii))/10;
        Bsec(kk,ii)=B(jj,ii);
        for ll=1:10
            Bsec(kk+1,ii)=Bsec(kk,ii)+step;
            kk=kk+1;
        end
    end
end

% Initialse state storage matrices
q = zeros(nn,4);        % Quaternion states
wRota = zeros(nn,3);
ActuTorq = zeros(nn,3);
MbDisplay = zeros(nn,3);
x = zeros(nn,3);
y = zeros(nn,3);
z = zeros(nn,3);
```

```
MagSen = zeros(nn,3);%instance magnitometer reading for the x axis
b1Cal=zeros(1,3);
b2Cal=zeros(1,3);
b3Cal=zeros(1,3);
b1norm=zeros(1,3);
b2norm=zeros(1,3);
b3norm=zeros(1,3);

d21=zeros(1,3);
d32=zeros(1,3);
Ampd21=zeros(1000,1);
Ampd32=zeros(1000,1);
% AmpDiff=zeros(2000,1);
CoilOn=zeros(nn,1);
SpeedTemp = 0;
% normMagCalRotaAxis = zeros(nn,3);

t = zeros(nn,1);        % Time points
%% Start the Computation for Visualization ???
tlimD = 1;      % Time limit per orbit step
dataReady = 0;
kk=1;
DataState = 0;
ActionStep = 1;
% EnergiseStep = 1;
DefaultStep = 1; % one step represents 1 second
% TorqK = 2;
DetumSys = 1;
% tempMeasureStep = 0;
% ll=1;
M = [0 0 0];
for jj = 1:nn %% Main Loop


    MbDisplay(jj,:)=M;

    if jj==1
        q(jj,:) = qo;%record the initial attitude
        qTemp=q(jj,:);
        wRota(jj,:)=wbo;%record the initial rotaional speed
        wTemp=wRota(jj,:);
    else
        B1 = Bsec(jj-1,1:3);
        B2 = Bsec(jj,1:3);
        [Mb,qn,wbn,Tg,tfin] =
LaDetum_Rotdyn(qTemp,wTemp,B1',B2',M,tlimD);%similar funtion to the
Simulink model
        q(jj,:) = qn;%record the attitude
        qTemp = qn;%feed the attitude back
        ActuTorq(jj,:) = Tg;
        wRota(jj,:)=wbn;%record the rotational speed
        wTemp = wbn;%feed the rotational speed back
    end

    if jj==1
        t(jj)=0;
    else
        t(jj) = t(jj-1) + tfin;
    end
    tlimD = 1*jj - t(jj);

    [x(jj,:),y(jj,:),z(jj,:)] = Q_2_Carti(q(jj,:));
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%
    % Jon's De-Tumble sensing
    % Calculate the rotational axis and the angle, based on the
magnitometor reading
    MagSen(jj,1) = dot(Bsec(jj,1:3),x(jj,:));%get the instance magnitometor
reading
    MagSen(jj,2) = dot(Bsec(jj,1:3),y(jj,:));
    MagSen(jj,3) = dot(Bsec(jj,1:3),z(jj,:));

    indexTemp(jj,1) = ActionStep;
    ActionStep = ActionStep - 1;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Gather the magnitometor data
    if ActionStep==0 && DataState==0
        b1Cal(1,1:3)=MagSen(jj,1:3);
        b1norm(1,1:3)=b1Cal(1,1:3)/norm(b1Cal(1,1:3));
        DataState = 1;
        ActionStep = DefaultStep;
        %          dataReady = 0;
    elseif ActionStep==0 && DataState==1
        b2Cal(1,1:3)=MagSen(jj,1:3);
        b2norm(1,1:3)=b2Cal(1,1:3)/norm(b2Cal(1,1:3));
        DataState = 2;
        ActionStep = DefaultStep;
        %          dataReady = 0;
    elseif ActionStep==0 && DataState==2
        b3Cal(1,1:3)=MagSen(jj,1:3);
        b3norm(1,1:3)=b3Cal(1,1:3)/norm(b3Cal(1,1:3));
        DataState = 3; % finish gathering data, go to analysis them
        ActionStep = DefaultStep;
        %          dataReady = 1;
    end

    if DataState==3 % analysis the data set that had just been collected
        %          dataReady=0;
        d21(1,1:3)=b2norm(1,1:3)-b1norm(1,1:3);
        d32(1,1:3)=b3norm(1,1:3)-b2norm(1,1:3);
        Ampd21(kk,1)=norm(d21(1,1:3));
        Ampd32(kk,1)=norm(d32(1,1:3));


theta(kk,1)=acos(dot(d21(1,1:3),d32(1,1:3))/(Ampd21(kk,1)*Ampd32(kk,1)));

alhpa(kk,1)=asin(((Ampd21(kk,1)+Ampd32(kk,1))/4)/sin(theta(kk,1)/2));

        RotaSpeed(kk,1) = theta(kk,1)/DefaultStep;% rad/sec measurement
        SpeedTemp = RotaSpeed(kk,1);

        StepAngle(kk,1) = pi/10;
        OptiStep = (StepAngle(kk,1))/RotaSpeed(kk,1); % how many steps
required to achieve about 45 degree
        OptiStep = ceil(OptiStep);

        if RotaSpeed(kk,1)<=0.01
            DetumSys = 0;
        else
```

```
            DetumSys = 1;
        end

        if alhpa(kk,1)>1.3963 % when the earth field is about perpendicular
to the rota axis
            if DetumSys == 1 % if the system is on
                if RotaSpeed(kk,1)<=0.05
                    TorqK = 0.8;
                elseif RotaSpeed(kk,1)<=1.2
                    TorqK = 1.554;
                end
                CoilOn(jj,1)=1; % coils on
                DataState = 4;   % go to the actuation mode
                %                 TorqK = 2;
            else
                DataState = 0;
                kk=kk+1;
            end
        else
            DataState=0;
            M = [0 0 0];
            kk=kk+1;
        end

    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%
    % Jon's De-Tumble control

    if ActionStep==0 && DataState==4 % as long as DataState stays >= 4, it
wouldn't go back to measure mode

        %          M = TorqK*RotaSpeed(kk,1)*b2norm(1,1:3);
        M = TorqK*b2norm(1,1:3);
        DataState = 5;
        ActionStep = DefaultStep;

    elseif ActionStep==0 && DataState==5

        %          M = TorqK*RotaSpeed(kk,1)*b2norm(1,1:3);
        M = TorqK*b2norm(1,1:3);
        DataState = 6;
        ActionStep = DefaultStep;

    elseif ActionStep==0 && DataState==6

        M = [0 0 0];

        DefaultStep = OptiStep;

        DataState = 0;
        ActionStep = DefaultStep;
        kk=kk+1;

    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    SpeedShow(jj,1) = SpeedTemp;
end
%% Controlor proformance analysis
% AvgSpeed = mean(theta)/DefaultStep
AvgSpeed = mean(SpeedShow(3:29991,1))
LowestSpeed = min(SpeedShow(3:29991,1))
for jj=1:nn
    normTorque(jj,1) = norm(ActuTorq(jj,:));

end
normMoment = zeros(nn,1);
normField = zeros(nn,1);
TorqueEffy = zeros(nn,1);
ActuAngle = zeros(nn,1);
AngleEffy = zeros(nn,2);

kk=1;
for jj=1:nn
%     normMoment(jj,1) = norm(MbDisplay(jj,:));%magnetic moment generated
by the coils
%     normField(jj,1) = norm(Bsec(jj,1:3));
    if norm(MbDisplay(jj,:))~=0
        TorqueEffy(jj,1) =
norm(ActuTorq(jj,:))/(norm(Bsec(jj,1:3))*norm(MbDisplay(jj,:)));
        TorqueEffyTrack(kk,1)=TorqueEffy(jj,1);
        ActuAngle(jj,1) =
180*acos( dot(MbDisplay(jj,:),Bsec(jj,1:3))/(norm(Bsec(jj,1:3))*norm(MbDisp
lay(jj,:)))  )/pi;
        AngleEffy(jj,1) = ActuAngle(jj,1);
        AngleEffy(jj,2) = TorqueEffy(jj,1);
        ActuAngleTrack(kk,1) = ActuAngle(jj,1);
        kk=kk+1;
    end
end
average=mean(TorqueEffyTrack(:,1))
minEff = min(TorqueEffyTrack(:,1))

%% Plot motion of satellite axes
if Displayrun==1
    figure(1)
    % if pic==2 %   Plot movie of satellite
    for jj = 1:nn
        v = [x(jj,:); y(jj,:); -x(jj,:); -y(jj,:)];
        fill3(v(:,1),v(:,2),v(:,3),'y')
        axis([-2 2 -2 2 -1.5 1.5])
        hold on
        plot3([0;x(jj,1)],[0;x(jj,2)],[0;x(jj,3)],'r')
        text(x(jj,1),x(jj,2),x(jj,3),'X','color','b','FontSize',16)
        plot3([0;y(jj,1)],[0;y(jj,2)],[0;y(jj,3)],'r')
        text(y(jj,1),y(jj,2),y(jj,3),'Y+Navi','color','b','FontSize',16)
        plot3([0;z(jj,1)],[0;z(jj,2)],[0;z(jj,3)],'r')
        text(z(jj,1),z(jj,2),z(jj,3),'Z','color','b','FontSize',16)

plot3([0;1.5*Bsec(jj,1)],[0;1.5*Bsec(jj,2)],[0;1.5*Bsec(jj,3)],'b')%plottin
g the earth magnetic field
        if norm(MbDisplay(jj,:))~=0

plot3([0;MbDisplay(jj,1)],[0;MbDisplay(jj,2)],[0;MbDisplay(jj,3)],'b')

text(MbDisplay(jj,1),MbDisplay(jj,2),MbDisplay(jj,3),'MagMo','color','b','F
ontSize',16)
```

```
        end

        view(0,0)
        title('Target:black    Mag Fld:blue    Down:cyan       Depress
space bar to step display')
        hold off
        Minutes = t(jj)/60
        pause
    end
end
```

LaDetum_Rotdyn.m

```
function [Mb,qn,wbn,Tb,tfin] = LaDetum_Rotdyn(qo,wbo,B1,B2,M,tlim)
% Calculates rotation of Satellite in a Mag Fld.
% Difference from RotdynG,   Input: M  inertial ref Mag Moment.
% qo, wbo : initial state of satellite
% qn, wbn : final state of satellite
% tlim: time to stop calculation
% tfin: final time of calculation

% Initialise satellite parameters
Inertia = [0.115 0.115 0.115]';     % Pr Mmts of Inertia of satellite
around CG. [0.2 0.2 0.05]
k = zeros(3,1);                 % Normalised inertia coefficients
k(1) = (Inertia(2)-Inertia(3))/Inertia(1);
k(2) = (Inertia(3)-Inertia(1))/Inertia(2);
k(3) = (Inertia(1)-Inertia(2))/Inertia(3);
Tb = zeros(3,1);            % Torque on satellite relative to body axes
A = zeros(3,3);         % Attitude matrix

% Initialise Attitude matrix
qm = qo;
A(1,:) = [qm(1)^2-qm(2)^2-qm(3)^2+qm(4)^2 2*(qm(1)*qm(2)+qm(3)*qm(4))
2*(qm(1)*qm(3)-qm(2)*qm(4))];
A(2,:) = [2*(qm(1)*qm(2)-qm(3)*qm(4)) -qm(1)^2+qm(2)^2-qm(3)^2+qm(4)^2
2*(qm(2)*qm(3)+qm(1)*qm(4))];
A(3,:) = [2*(qm(1)*qm(3)+qm(2)*qm(4)) 2*(qm(2)*qm(3)-qm(1)*qm(4)) -qm(1)^2-
qm(2)^2+qm(3)^2+qm(4)^2];

% Initialse computation
tlimh = tlim/2;
tfin = 0;                       % Rotation starts at time zero
wbn=wbo; qn=qo;  % Initialise state for 1st iteration step
% Start computation
while tfin < tlim
    if tfin < tlimh
        B=B1;
    else B=B2;
    end
    wbl=wbn; wbm=wbn; wbnn=wbn; ql=qn; qm=qn; qnn=qn;   % Initialise state
for next integration step
    %  Ensure the step size is no greater than 0.01 rad = 0.57 deg
    if norm(wbm) <= 0.01
        dt=1;
    else dt = 0.01/norm(wbm);
    end
    f1=0;   f2=0;   iter = 1;       % Reset convergence flags and counter
    % Start iteration loop for next time step
    while (f1*f2|iter>9)==0
%         Mb = A*M';        % Dipole moment relative to body axes
        Mb = M';
        Bb = A*B;       % Mag field relative to body axes
        Tb(1) = Mb(2)*Bb(3)-Mb(3)*Bb(2);    % Calculate torque
        Tb(2) = Mb(3)*Bb(1)-Mb(1)*Bb(3);
        Tb(3) = Mb(1)*Bb(2)-Mb(2)*Bb(1);
        wbn(1) = wbl(1) + (k(1)*wbm(2)*wbm(3) + Tb(1)/Inertia(1))*dt;
        wbn(2) = wbl(2) + (k(2)*wbm(3)*wbm(1) + Tb(2)/Inertia(2))*dt;
        wbn(3) = wbl(3) + (k(3)*wbm(1)*wbm(2) + Tb(3)/Inertia(3))*dt;
        wbm = (wbl+wbn)/2;
        if norm(wbn-wbnn)<0.00000001
            f1=1;
        else f1=0;
        end
        wbnn = wbn;         % Retain for next iteration
        Q = [0 wbm(3) -wbm(2) wbm(1); -wbm(3) 0 wbm(1) wbm(2);
            wbm(2) -wbm(1) 0 wbm(3); -wbm(1) -wbm(2) -wbm(3) 0];
        qn = ql + 0.5*qm*Q'*dt;
        qn = qn/norm(qn);
        qm = (ql+qn)/2;
        if norm(qn-qnn) < 0.000001
            f2=1;
        else f2=0;
        end
        qnn = qn;           % Retain for next iteration
        A(1,:) = [qm(1)^2-qm(2)^2-qm(3)^2+qm(4)^2
2*(qm(1)*qm(2)+qm(3)*qm(4)) 2*(qm(1)*qm(3)-qm(2)*qm(4))];
        A(2,:) = [2*(qm(1)*qm(2)-qm(3)*qm(4)) -qm(1)^2+qm(2)^2-
qm(3)^2+qm(4)^2 2*(qm(2)*qm(3)+qm(1)*qm(4))];
        A(3,:) = [2*(qm(1)*qm(3)+qm(2)*qm(4)) 2*(qm(2)*qm(3)-qm(1)*qm(4)) -
qm(1)^2-qm(2)^2+qm(3)^2+qm(4)^2];
        iter = iter+1;
        if iter == 10  tfin, else, end      % Signal if convergence failure
    end
    tfin = tfin + dt;
end
```

Q_2_Carti.m

```
function [ xTemp,yTemp,zTemp ] = Q_2_Carti( qm )
% this function will receive a quaternion and convert it back to Cartesian
x,y,z coordinate
     xTemp(1,:) = [qm(1)^2-qm(2)^2-qm(3)^2+qm(4)^2
2*(qm(1)*qm(2)+qm(3)*qm(4)) 2*(qm(1)*qm(3)-qm(2)*qm(4))];
    yTemp(1,:) = [2*(qm(1)*qm(2)-qm(3)*qm(4)) -qm(1)^2+qm(2)^2-
qm(3)^2+qm(4)^2 2*(qm(2)*qm(3)+qm(1)*qm(4))];
    zTemp(1,:) = [2*(qm(1)*qm(3)+qm(2)*qm(4)) 2*(qm(2)*qm(3)-qm(1)*qm(4)) -
qm(1)^2-qm(2)^2+qm(3)^2+qm(4)^2];
end
```

LaDetum_GraphcaDisplay3_CountDown.m

```
nn = 29991;
step = floor(nn/12);

filename = 'Detum_GraphicalData.txt';
fid = fopen(filename,'W');

filename2 = 'Detum_ShortGraphicalData.txt';
fid2 = fopen(filename2,'W');

filename3 = 'Detum_SmoothGraphicalData.txt';
fid3 = fopen(filename3,'W');

ii = 0;
PreY = NaN;
PreP = NaN;
PreR = NaN;
SmoothCount = 0;
Hour = int32(3600);
Minit = int32(60);

for jj = 1:nn
    pitch = asin(2*(q(jj,2)*q(jj,4)-q(jj,1)*q(jj,3)));
    roll = atan3((2*(q(jj,1)*q(jj,2)+q(jj,3)*q(jj,4)))/cos(pitch),(1-
2*(q(jj,2)*q(jj,2)+q(jj,3)*q(jj,3)))/cos(pitch));
    yaw = atan3((1-
2*(q(jj,1)*q(jj,1)+q(jj,2)*q(jj,2)))/cos(pitch),(2*(q(jj,2)*q(jj,3)+q(jj,1)
*q(jj,4)))/(cos(pitch)));

    if yaw*180/pi<0
        Yp = yaw*180/pi+360;
    else
        Yp = yaw*180/pi;
    end

    if pitch*180/pi<0
        Pp = pitch*180/pi+360;
    else
        Pp = pitch*180/pi;
    end

    if roll*180/pi<0
        Rp = roll*180/pi+360;
    else
        Rp = roll*180/pi;
    end

    Yp=round(Yp);
    Pp=round(Pp);
    Rp=round(Rp);

    hour = idivide(nn-jj,Hour);
    minit = idivide(nn-jj-hour*3600,Minit);
    secound = nn-jj-hour*3600-minit*60;

    %getting the number to the desired string to print on to the file
    file_string = Detum_nums2str2(Yp,Pp,Rp,hour,minit,secound);
    fprintf(fid,file_string);
    if( (jj <= (step*ii+100)) && (jj>step*ii) )
        fprintf(fid2,file_string);

        if ~isnan(PreY) && ~isnan(PreP) && ~isnan(PreR)
            for SmoothCount = 0:10
                ratio = SmoothCount/10;
                num1 = PreY*(1-ratio)+ratio*Yp;
                num2 = PreP*(1-ratio)+ratio*Pp;
                num3 = PreR*(1-ratio)+ratio*Rp;
                if num1>360
                    num1 = num1-360;
                end
                if num2>360
                    num2 = num2-360;
                end
                if num3>360
                    num3 = num3-360;
                end
                num1 = round(num1);
                num2 = round(num2);
                num3 = round(num3);
                file_string2 = Detum_nums2str( num1, num2, num3);
                fprintf(fid3,file_string2);
            end
        end
        PreY = Yp;
        PreP = Pp;
        PreR = Rp;
    elseif(jj == (step*ii+101))
        ii = ii+1;
    end
end

fclose(fid);
fclose(fid2);
fclose(fid3);
```

Detum_num2str2.m

```
function String_4_file = Detum_nums2str2( num1,num2,num3,num4,num5,num6)
%UNTITLED3 Summary of this function goes here
%   Detailed explanation goes here
    number_length = zeros(1,6);
    number_length(1,1) = numel(num2str(num1));
    number_length(1,2) = numel(num2str(num2));
    number_length(1,3) = numel(num2str(num3));
    number_length(1,4) = numel(num2str(num4));
    number_length(1,5) = numel(num2str(num5));
    number_length(1,6) = numel(num2str(num6));
%     temp_data(1,1)=Yp;
%     temp_data(1,2)=Pp;
%     temp_data(1,3)=Rp;
    num_str1 = blanks(6);
    num_str2 = blanks(6);
    num_str3 = blanks(6);
    num_str4 = blanks(6);
    num_str5 = blanks(6);
    num_str6 = blanks(6);
    num_str1(6-number_length(1,1)+1:6) = num2str(num1);
    num_str1 = strcat(num_str1,'\t');
    num_str2(6-number_length(1,2)+1:6) = num2str(num2);
    num_str2 = strcat(num_str2,'\t');
    num_str3(6-number_length(1,3)+1:6) = num2str(num3);
    num_str3 = strcat(num_str3,'\t');
    num_str4(6-number_length(1,4)+1:6) = num2str(num4);
    num_str4 = strcat(num_str4,'\t');
    num_str5(6-number_length(1,5)+1:6) = num2str(num5);
    num_str5 = strcat(num_str5,'\t');
    num_str6(6-number_length(1,6)+1:6) = num2str(num6);
    num_str6 = strcat(num_str6,'\r\n');

    String_4_file =
strcat(num_str1,num_str2,num_str3,num_str4,num_str5,num_str6);

end
```

La_DetumControll.c

```
# include <stdio.h>
# include <math.h>
# define pi 3.141592653589793
#include "coils.h"
int main(){

       int ActionStep = 0;
       float MagSen[3][3];
       float MagNorm[3];
       float MagPostNorm[3][3];
       float *tempArry;
       long int ProcessTime = 0;
       long int StepTime = 1, newStepTime;
       bool ControlSystemOn = 0;

       float d21[3],d32[3];
       float Norm_d21,Norm_d32;
       float theta,alpha;
       float InstanceSpeed;//unit in rad/second
       float StepAngle = pi/10;//unit in rad
       float TorqueK = 0;

       float ActuateMoment[3];
       float CoilX;
       float CoilY;
       float CoilZ;
       coilSwitch pcs[MAX_STATES];
       int CoilStates = 0;

       while(true){
                     //step 1, take magnetometer reading
                     if(ActionStep == 0 && mills()-ProcessTime>StepTime){//need to
know the real-time here (assumed function call "mills()")
                                   ProcessTime = mills();//keep track of what is the
time now (update the thing)
                                   tempArry = ReadMag();//need to get the reading from
the magnetometer here
                                   MagSen[0][0] = tempArray->0;
                                   MagSen[0][1] = tempArray->1;
                                   MagSen[0][2] = tempArray->2;
                                   //normalize the reading
                                   MagNorm[0] = 
norm(MagSen[0,0],MagSen[0,1],MagSen[0,2]);//require meth function "norm()"

                                   MagPostNorm[0][0] = MagSen[0][0]/MagNorm[0];
                                   MagPostNorm[0][1] = MagSen[0][1]/MagNorm[0];
                                   MagPostNorm[0][2] = MagSen[0][2]/MagNorm[0];
                                   ActionStep = 1;//increment to next step
                            }
                     if(ActionStep == 1 && mills()-ProcessTime>StepTime){//need to 
know the real-time here (assumed function call "mills()")
                                   ProcessTime = mills();//keep track of what is the 
time now (update the thing)
                                   tempArry = ReadMag();//need to get the reading from 
the magnetometer here
                                   MagSen[1][0] = tempArray->0;
                                   MagSen[1][1] = tempArray->1;
                                   MagSen[1][2] = tempArray->2;
                                   //normalize the reading
                                   MagNorm[1] = 
norm(MagSen[1,0],MagSen[1,1],MagSen[1,2]);//require meth function "norm()"

                                   MagPostNorm[1][0] = MagSen[1][0]/MagNorm[1];
                                   MagPostNorm[1][1] = MagSen[1][1]/MagNorm[1];
                                   MagPostNorm[1][2] = MagSen[1][2]/MagNorm[1];
                                   ActionStep = 2;//increment to next step
                            }
                     if(ActionStep == 2 && mills()-ProcessTime>StepTime){//need to 
know the real-time here (assumed function call "mills()")
                                   ProcessTime = mills();//keep track of what is the 
time now (update the thing)
                                   tempArry = ReadMag();//need to get the reading from 
the magnetometer here
                                   MagSen[2][0] = tempArray->0;
                                   MagSen[2][1] = tempArray->1;
                                   MagSen[2][2] = tempArray->2;
                                   //normalize the reading
                                   MagNorm[2] = 
norm(MagSen[2,0],MagSen[2,1],MagSen[2,2]);//require math function "norm()"

                                   MagPostNorm[2][0] = MagSen[2][0]/MagNorm[2];
                                   MagPostNorm[2][1] = MagSen[2][1]/MagNorm[2];
                                   MagPostNorm[2][2] = MagSen[2][2]/MagNorm[2];
                                   ActionStep = 3;//increment to next step
                            }
                     //preform calculation right after 3 data points are collected
                     if(ActionStep == 3){
                                   d21[0] = MagPostNorm[1][0] - MagPostNorm[0][0];
                                   d21[1] = MagPostNorm[1][1] - MagPostNorm[0][1];
                                   d21[2] = MagPostNorm[1][2] - MagPostNorm[0][2];

                                   d32[0] = MagPostNorm[2][0] - MagPostNorm[1][0];
                                   d32[1] = MagPostNorm[2][1] - MagPostNorm[1][1];
                                   d32[2] = MagPostNorm[2][2] - MagPostNorm[1][2];

                                   Norm_d21 = norm(d21[0],d21[1],d21[2]);//required 
norm()
                                   Norm_d32 = norm(d32[0],d32[1],d32[2]);

                                   theta = 
acos(dot(&d21,&d32)/(Norm_d21*Norm_d32));//required dot() and acos() functions
```

```
                    alpha = 
asin(((Norm_d21+Norm_d32)/4)/sin(theta/2));//required sin() and asin() functions

                    InstanceSpeed = theta/StepTime;

                    //adjust the step time
                    newStepTime = StepAngle/InstanceSpeed;

                    if(InstanceSpeed<0.01){//is the satellite stable 
enough yet?
                            ControlSystemOn = 0;
                            ActionStep = 0;
                            break;//break out of the while(true) 
loop
                        }else{
                                ControlSystemOn = 1;
                            }
                    if(alpha>1.3963){//is it a optimum angle?
                            if(ControlSystemOn == 1){
                                if(InstanceSpeed<0.05){
                                        TorqueK = 0.8;
                                    }

                                else if(InstanceSpeed<1.2){
                                        TorqueK = 1.554;
                                    }
                                ActionStep = 4;
                            }
                        }else{
                                ControlSystemOn = 0;//turn off 
the control system if it is not an optimum angle
                                ActionStep = 0;
                            }

                }
            if(ActionStep == 4 && mills()-ProcessTime>StepTime){
                    ProcessTime = mills();//keep track of what is the 
time now (update the thing)
                    ActuateMoment[0] = TorqueK*MagPostNorm[1][0];
                    ActuateMoment[1] = TorqueK*MagPostNorm[1][1];
                    ActuateMoment[2] = TorqueK*MagPostNorm[1][2];
                    CoilX = ActuateMoment[0]*StepTime*1000.0/1.554;
                    CoilY = ActuateMoment[1]*StepTime*1000.0/1.554;
                    CoilZ = ActuateMoment[2]*StepTime*1000.0/1.554;
                    CoilStates = 0;
                    if(CoilX > 0){
                        pcs[CoilStates].coils = IO2_COIL_XP;
                        pcs[CoilStates++].msec = CoilX;
                        CoilStates++;
                    }else if(CoilX < 0){
                        pcs[CoilStates].coils = IO2_COIL_XN;
                        pcs[CoilStates++].msec = CoilX;
                        CoilStates++;
                    }
                    if(CoilY > 0){
                        pcs[CoilStates].coils = IO2_COIL_YP;
                        pcs[CoilStates++].msec = CoilY;
                        CoilStates++;
                    }else if(CoilY < 0){
                        pcs[CoilStates].coils = IO2_COIL_YN;
                        pcs[CoilStates++].msec = CoilY;
                        CoilStates++;
```

```
                                   }
                                   if(CoilZ > 0){
                                          pcs[CoilStates].coils = IO2_COIL_ZP;
                                          pcs[CoilStates++].msec = CoilZ;
                                          CoilStates++;
                                   }else if(CoilZ < 0){
                                          pcs[CoilStates].coils = IO2_COIL_ZN;
                                          pcs[CoilStates++].msec = CoilZ;
                                          CoilStates++;
                                   }
                                   powerAdacCoils(pcs,CoilStates);
                                   //ActuateCoils(&ActuateMoment);//required the
function ActuateCoils() to actuate the coils
                                   ActionStep = 5;
                            }
                     if(ActionStep == 5 && mills()-ProcessTime>StepTime){
                                   ProcessTime = mills();//keep track of what is the
time now (update the thing)
                                   ActuateMoment[0] = TorqueK*MagPostNorm[1][0];
                                   ActuateMoment[1] = TorqueK*MagPostNorm[1][1];
                                   ActuateMoment[2] = TorqueK*MagPostNorm[1][2];
                                   CoilX = ActuateMoment[0]*StepTime*1000.0/1.554;
                                   CoilY = ActuateMoment[1]*StepTime*1000.0/1.554;
                                   CoilZ = ActuateMoment[2]*StepTime*1000.0/1.554;
                                   CoilStates = 0;
                                   if(CoilX > 0){
                                          pcs[CoilStates].coils = IO2_COIL_XP;
                                          pcs[CoilStates++].msec = CoilX;
                                          CoilStates++;
                                   }else if(CoilX < 0){
                                          pcs[CoilStates].coils = IO2_COIL_XN;
                                          pcs[CoilStates++].msec = CoilX;
                                          CoilStates++;
                                   }
                                   if(CoilY > 0){
                                          pcs[CoilStates].coils = IO2_COIL_YP;
                                          pcs[CoilStates++].msec = CoilY;
                                          CoilStates++;
                                   }else if(CoilY < 0){
                                          pcs[CoilStates].coils = IO2_COIL_YN;
                                          pcs[CoilStates++].msec = CoilY;
                                          CoilStates++;
                                   }
                                   if(CoilZ > 0){
                                          pcs[CoilStates].coils = IO2_COIL_ZP;
                                          pcs[CoilStates++].msec = CoilZ;
                                          CoilStates++;
                                   }else if(CoilZ < 0){
                                          pcs[CoilStates].coils = IO2_COIL_ZN;
                                          pcs[CoilStates++].msec = CoilZ;
                                          CoilStates++;
                                   }
                                   powerAdacCoils(pcs,CoilStates);
                                   //ActuateCoils(&ActuateMoment);//required the
function ActuateCoils() to actuate the coils
                                   ActionStep = 6;
                            }
                     if(ActionStep == 6 && mills()-ProcessTime>2*StepTime){
                                   ProcessTime = mills();//keep track of what is the
time now (update the thing)
                                   //ActuateMoment[0] = 0;
                                   //ActuateMoment[1] = 0;
                                   //ActuateMoment[2] = 0;
                                   CoilStates = 0;
                                   pcs[CoilStates].coils = 0;
                                   pcs[CoilStates++].msec = 0;
                                   powerAdacCoils(pcs,CoilStates);
                                   //ActuateCoils(&ActuateMoment);//required the
function ActuateCoils() to actuate the coils

                                   StepTime = newStepTime;//renew the StepTime based
on "updated" speed measurement
                                   ActionStep = 0;
                            }

              }
}
```