Low cost magnetic simulator for attitude control

Mario Castro Santiago[1], *Luis Sánchez Velasco*[2]
GranaSAT Aerospace Group University of Granada
Granada, Spain
[1]mario.cs.96.az@gmail.com, [2]hfegetude@gmail.com

Andrés Roldán Aranda
Electronics Department
University of Granada
Granada, Spain
amroldan@ugr.es

***Abstract*—The aim of this paper is to propose an aerospace project, in order to allow Master's Space students to put in practice their knowledge in physics, which can be crucial in this engineering area. Here, we face the attitude control issue of a cubesat, using a physical 1U satellite model and a magnetic field simulator, combined with real data from a desired orbit. The satellite model will use magnetorquers controlled by an Arduino board, in the same way as the magnetic simulator.**

Keywords—Aerospace application; magnetic field simulator; attitude control; cubesat magnetorquer

I. INTRODUCTION

Nowadays, Physics knowledge can play an important role in the aerospace environment, since it provides an overall vision of many aspects in a space mission. For instance, reduce the roll rate in a satellite after its orbital insertion is a necessary step for any mission. One can achieve this by using magnetic actuators, called magnetorquers, which are a low cost and mass solution, particularly in small cubesats at LEO (Low Earth Orbit).

In point of fact, the problem faced here would be part of the so-called ADCS (Attitude Determination and Control System) of a satellite, a very relevant part of Aerospace Engineering since, even there are some space mission that do not require neither spacecraft active stabilization nor orientation, this is not the case of many satellites which need a certain degree of pointing. For instance, the usage of directional antennas, cameras or some sensors, forces engineers to undertake this issue. Although there exist plenty of literature about this, some crucial references are [1], [2] and [3].

A unique satellite manufacturing standard made it appearance in 1999 by the hand of California Polytechnic State University, San Luis Obispo and Stanford University's Space Systems Development Lab, in order to ease access to space by university students [4]. This was called Cubesat. As its name suggest, Cubesats are modular satellites made of 10x10x10 cm cubes. Depending of the number of cubes, they are called 1U, 2U, 6U… being U the 1000 cm^3 volume unit.

Here, we will provide methods for testing magnetorquers and control algorithms, using a simple magnetic simulator [5] and a 1U cubesat prototype, both controlled by different Arduino ATMega2560. In addition, the cubesat model will contain an inertial measurement unit (IMU), which provides real-time attitude information to its Arduino board (also called On-Board Computer, OBC). All scripts will be written in C++, taking into account the memory limit of Arduino Mega boards.

II. PROJECT GENERAL APPROACH

Although his project comprises many steps, they may be all included in two groups: simulation and physical implementation.

A. Simulation

Simulation tasks are the basis of the whole project. They are not only the framework of the expected testing results, they also allow us to design, configure or improve control algorithms. Two things have to be modeled: a satellite and an orbit around Earth. The satellite attitude and position will be simulated both in MATLAB and AGI STK (System Tool Kit) [6], a widely used Aerospace Software which will validate some aspects of our simulation.

Concerning the satellite, a 3D model must be provided to STK, which is discussed in Section III. Since magnetorquers are used as actuators, we should define too their properties (in Section VIII, some values are given). Providing a control algorithm is crucial too: B-dot algorithm [7] is a simple way of stabilization, and that why it is the chosen here, as a first step of ADCS designing (Section V). When it comes to orbit simulation, we have to propagate the satellite's trajectory and compute the magnetic field existing in that orbit (Section IV).

B. Physical implementation

This part involves two tasks: sensor calibration and porotype testing. In this case we only calibrate the magnetometers of the IMU, and this process is described in Section VII. Finally, having building a prototype, it can be tested inside the magnetic simulator. The satellite must send information about its angular speed to a computer in order to compare with graphs obtained in simulations.

Fig.1 schematizes the general process.

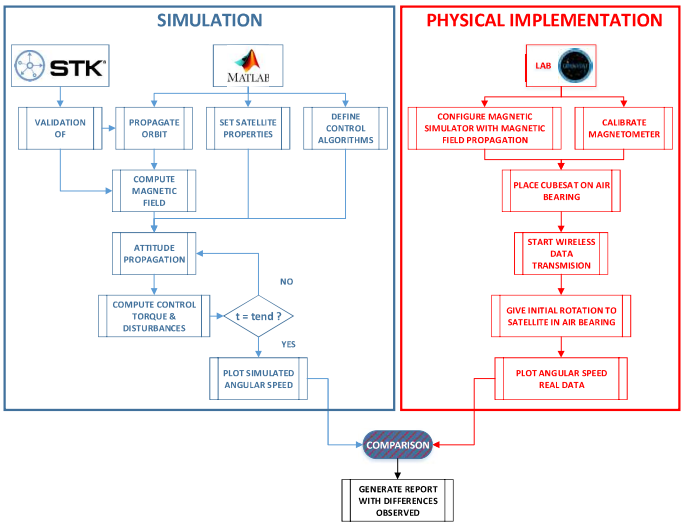


Fig.1: Model treatment in Blender

III. DESIGN OF A 3D CUBESAT MODEL

A. *General proceedings*

The first global part of this project consists on modeling a 3D model of our cubesat, in order to use it in STK software. The whole process is summarized in Fig.2, and a manual is found in [16].

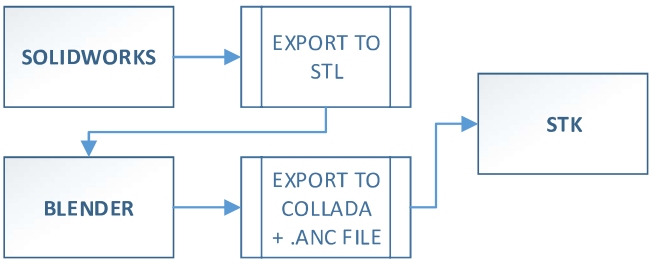


Fig.2 Flow chart of the design process

B. *Cubesat design in SolidWorks*

First of all, we need to create our model in *SolidWorks*. This model should be as simple as possible: we do not need screws, batteries, circuits, etc. If you are using a CAD file with a real satellite, with all those internal parts, they must be disabled or removed. After this process, we export a STL file with our model, just containing the "skeleton" and solar panels.

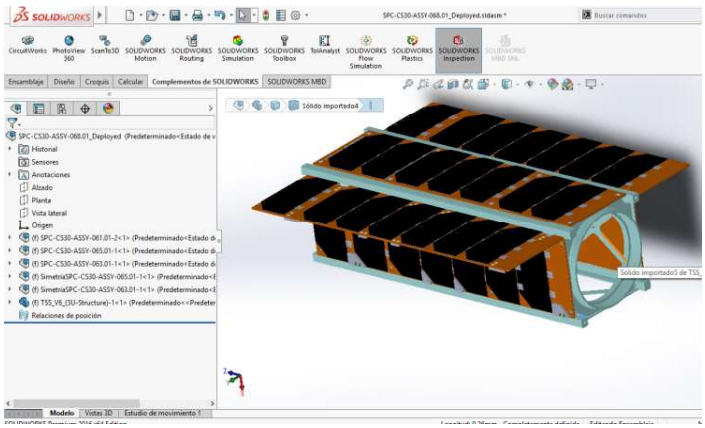


Fig.3: Cubesat modeling in SoliWorks

C. *Creating COLLADA model by Blender*

Our STL file is not ready for using it in STK yet. We need to group the cubesat parts in at least two types: solar panels and the rest. It is important, since later, STK will need to know which parts of the model (specifically, their area and solar efficiency) are photovoltaic panels. Moreover, our models can have mobile parts, called "articulations", e.g, foldable panels, like the ones shown in Fig.3. After grouping all these parts in "layers" in Blender, Fig.4, we can export a COLLADA (.dae) file. Before loading it in STK, an "ancillary" file (.anc) must be written, containing information about degrees of freedom each part has for motion, sensor attachment points, pointing directions of moving parts.

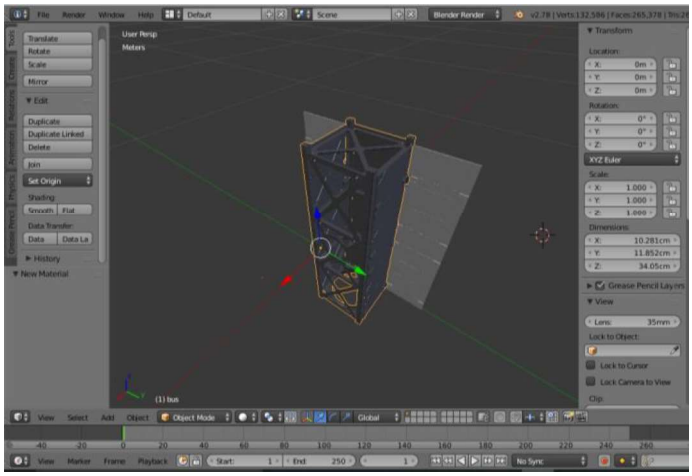


Fig.4: Model treatment in Blender

D. *Loading model in STK*

After generating a .dae file, and writing a .anc file, we can load our model in STK, Fig.5, which will recognize solar panels and articulations, as we have defined them before.

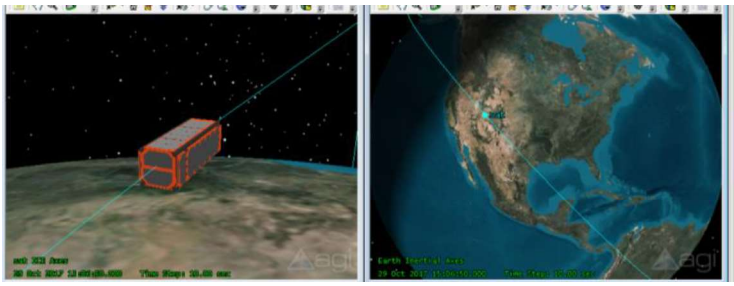


Fig.5: Model loaded in STK

IV. PROPAGATING ORBIT AND MAG. FIELD

In order to generate a real magnetic field vector inside the simulator, the orbital trajectory of the satellite must be propagated. There are several algorithms to achieve this; here, SGP4 [8] is implemented in C++. We have write it both in the simulator board and in the OBC, since in future steps, the satellite will need to estimate its position in order to improve attitude control by the use of Kalman filter. Thereby, simulator receives real orbital position, so it can compute magnetic field by the use of IGRF [9] model, which estimates it in some given coordinates in ECI (Earth-Centered Inertial) frame, i.e a reference frame with its center locater in Earth origin and does not rotate. Finally, Helmholtz coils generates this magnetic vector. This process is summarized in Fig.6.

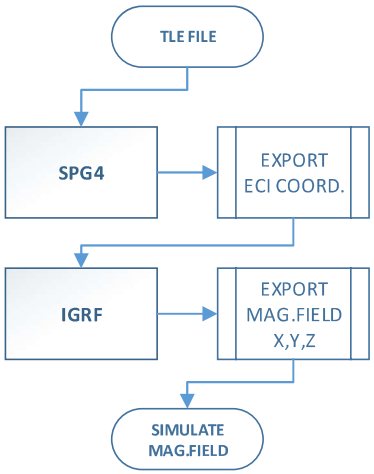


Fig.6: Propagation process flow chart

A. SGP4 implementation

The input necessary when propagating a spacecraft orbit is called TLE (Two eLement Set). It is a text document which contains a list of orbital elements in a specific time (epoch). The SGP4 algorithm (code in [10]) uses these data to estimate (both in the past and future) the trajectory of de satellite. In order to validate the effectivity of our SGP4 implementation, we propagated ISS trajectory both in STK and MATLAB. Results are shown in Fig.7a-b. The small differences between both propagations allow us finally write SGP4 algorithm in C++.

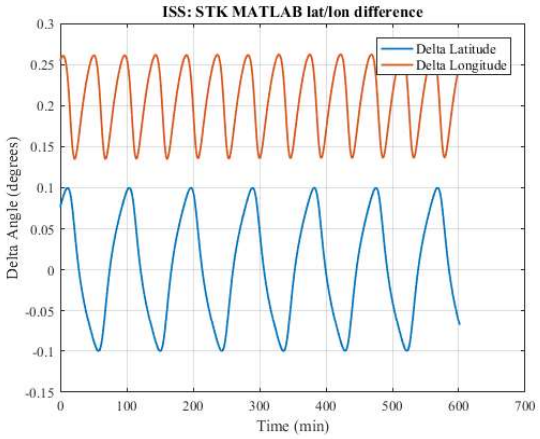


Fig.7a: STK versus MATLAB propagated lat/lon difference

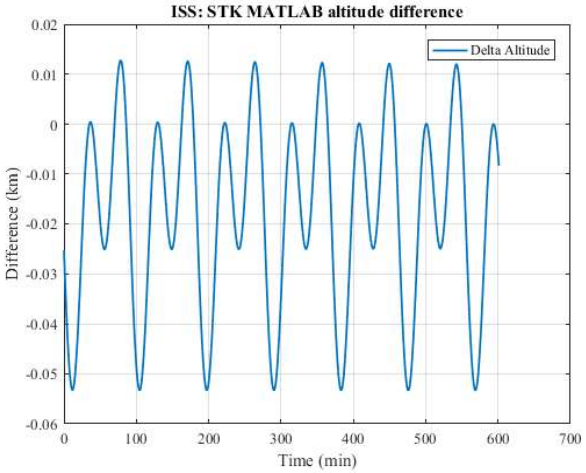


Fig.7b: STK versus MATLAB propagated altitude difference

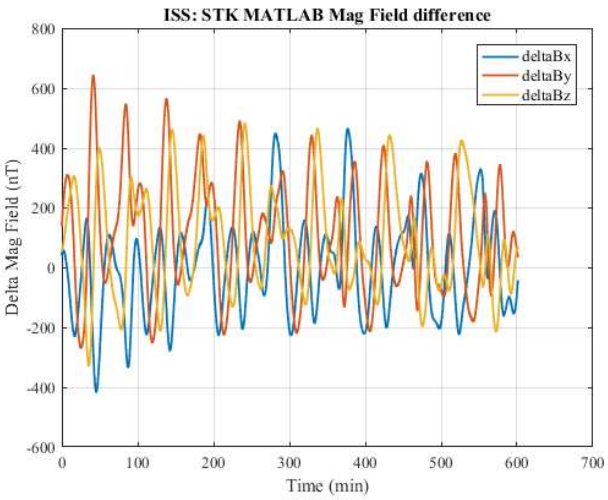


Fig.8: STK versus MATLAB propagated magnetic field difference

B. IGRF implementation

Next step consists on compute magnetic field in spatial points propagated by SGP4. Again, we first wrote MATLAB code for testing this algorithm (code can be found in [11]), and then compared it with STK magnetic field estimation. Results are shown in Fig.8. Finally, IGRF program was written in C++.

V. ATTITUDE CONTROL ALGORITHM

Although there exist many control algorithms, here we have implemented the well-known B-dot algorithm. It is briefly described in this section, as well as its implementation.

A. B-dot: physics

The effectivity of this method is based on the fact that magnetic field time derivative vector, i.e, the rate change of 3 magnetic field components, can be written as:

$$\dot{\boldsymbol{B}} \approx \boldsymbol{\omega} \times \boldsymbol{B} \qquad (1)$$

Where $\dot{\boldsymbol{B}}$ is the derivative vector mentioned before, $\boldsymbol{\omega}$ is the angular velocity vector of the satellite, and $\boldsymbol{B}$ is just the magnetic vector; all of them expressed in satellite body frame coordinates. So, if we need to reduce our angular speed, we need to create torque in its direction, but with opposite sense. This can be achieved by generating a magnetic moment vector ($\boldsymbol{m}$) in the form:

$$\boldsymbol{m} = -\alpha\ \dot{\boldsymbol{B}} \qquad (2)$$

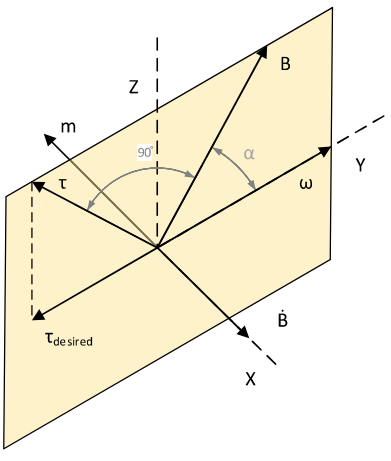


*Fig.9: All vectors are in the same plane (ZY) except **m** and $\dot{\boldsymbol{B}}$*

Where α is just a constant. Taking into account the expression of the torque (***T***) created by a magnetic dipole:

$$\boldsymbol{T} = \boldsymbol{m} \times \boldsymbol{B} \quad (3)$$

Then, generating ***m*** with magnetorquers, we are creating a component of torque which has equal direction as ***ω*** but opposite sense: we are reducing the angular speed rate. The relation between vectors named here can be seen in Fig.9

B. *Implementation*

The B-dot algorithm implies that we have to compute the derivative of the magnetic field. We can estimate it with consecutive measurements of ***B(t),*** and just applying the derivation concept:

$$\dot{\boldsymbol{B}} = [\boldsymbol{B}(t+dt) - \boldsymbol{B}(t)]/dt \quad (4)$$

Where *dt* is the time between two measurements going to zero. Since we need to now the intensity values in each coil (actually, we use PWM for "change" current), we need to use equation (5):

$$\boldsymbol{I} = \boldsymbol{m}/(An) \quad (5)$$

Where *A* is the area of the coil, and *n* the number of turns. Finally, we can solve for intensity thanks to (2). The constant α works as a gain factor; here we use α = 1E-5. The OBC will carry on this algorithm, sending to the coil the right amount of intensity.

VI. PHYSICAL IMPLEMENTATION

A. *Testbed*

We make use of a testbed (Fig.10), which injects air to a hemisphere containing the cubesat model. This way, the hemisphere rates free, since there are not frictional forces. Places inside this device, the satellite model is located inside the magnetic simulator. Once the air is circulating, a slight touch to the satellite is enough for it to continue rolling indefinitely.

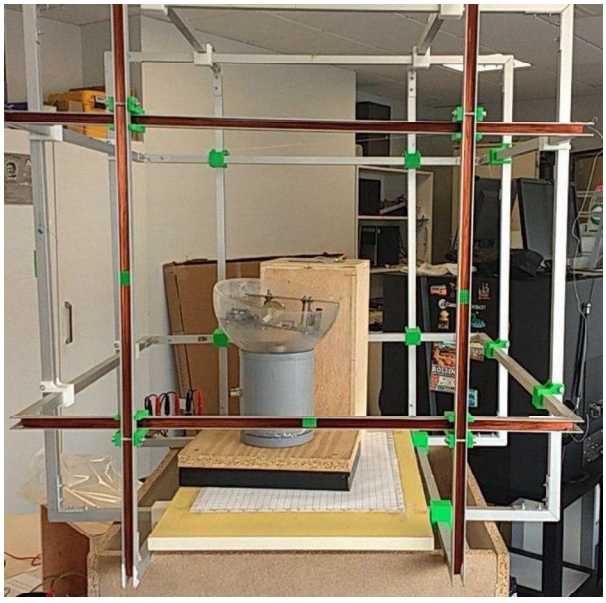
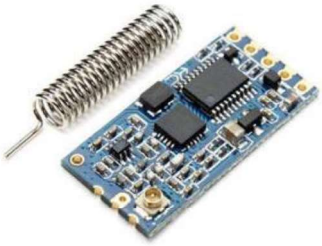


Fig.10: The testbed (grey cylinder) inside the mag. simulator

B. *Monitorizating variables*

Onboard sensors are able to measure 3D angular speed, acceleration, magnetic field and also temperature. These data (telemetry) are sent with the aid of a low cost commercial wireless serial module (Fig.11), directly from the OBC to an external computer. This allows us to know attitude parameters in real time.



α

Fig.11: Wireless serial module with antenna. (eBay)

α + β = γ (1) (1)

VII. MAGNETOMETER CALIBRATION

Before using the onboard magnetometer, it must be calibrated in order to get reliable measurements. Supposing we make a set of measurements with a correctly calibrated magnetometer, while we are rotating it along 3 spatial directions, we will be able to plot the magnitude of the local magnetic field in many directions, in a 3D graph. This set of points could be perfectly fitted by a sphere, with radius |***B***|, and centered in (0,0,0). However, commercial magnetometers are far away from this situation. Since we cannot modify them, we have to find a relation between the actual measurements, and the ideal ones. The process is briefly described in this section; fully geometrical interpretation and code can be found in [12].

A. *Getting raw data*

The first step consists on make a relative large number of magnetic field measurements in many directions. This is achieved by connecting the IMU to an Arduino board, and serial-writing the magnetometer output in x, y, z directions. Then, we just have to copy a large number of rows in the serial monitor and save them in a .txt file.

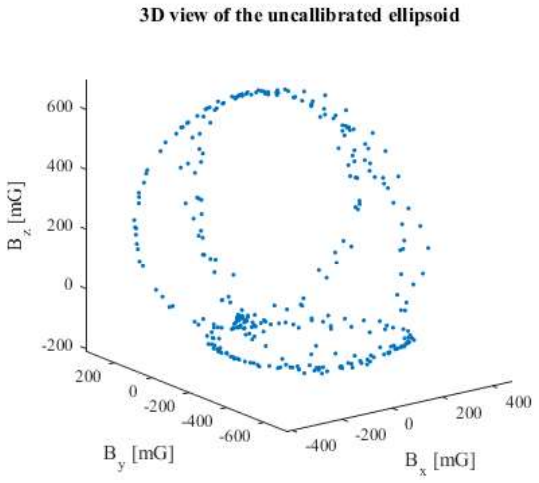


Fig.12: Raw magnetometer measurements

As shown in Fig.12, there are two basic issues: points do not form a sphere (it is an ellipsoid, this error is called soft iron), and they are not equally distributed around the origin

(there exists an offset, called hard iron). These differences with the ideal situation will be mathematically analyzed, and thus we will obtain some parameters with which raw measurements can be automatically transformed into more realistic magnetic field values.

B. Calibration parameters with MATLAB

Although is fully described in [11], the process followed in order to obtain calibration parameters is very intuitive. First, the location of the center of the ellipsoid is calculated, and this is essentially the offset (in three axis). The algorithm set the ellipsoid centered in that point. Then, the 3 principal axes of the ellipsoid are aligned with the x, y, z axis of the reference frame. Since they have different lengths, they must be scaled. This produces a sphere, which is then rotated so it has the original orientation of the ellipsoid. Finally, this sphere is scaled to a radius $|\boldsymbol{B}|$. The calibrated data is compared with the original ones in Fig.13. The hard iron effect is highly visible, since the centers of the both figures are obviously in different positions. The soft iron effect also appears in this graph: blue points form a sphere, red ones do not.

This process results in a 3 components vector $\boldsymbol{H_{cal}}$ (hard iron correction, i.e, the offset), and a 3x3 matrix $\boldsymbol{S_{cal}}$ (soft iron correction).

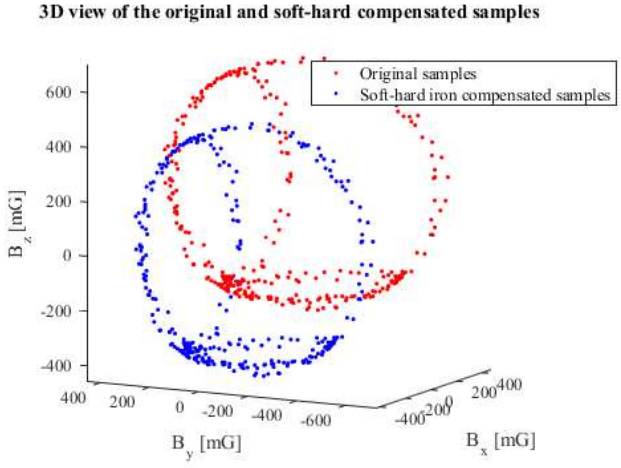


Fig.13: Calibrated (blue) and uncalibrated (red) points

C. Applying the calibration

Each time the cubesat measures the magnetic field, the raw measurements are transformed by the following relation:

$$\boldsymbol{B_{calibrated}} = \boldsymbol{S_{cal}} \cdot (\boldsymbol{B_{measured}} - \boldsymbol{H_{cal}}) \quad (6)$$

VIII. ATTITUDE CONTROL SIMULATION

Attitude control of the cubesat must be simulated via MATLAB before testing the prototype. In order to simulate the cubesat's attitude, we make use of a MATLAB toolbox called PROPAT [13], which integrates Euler's equations of motion, and takes into account external torque. This torque could be environmental (gravity, air drag…) or defined by the user, by a control law. As said before, we have implemented B-dot algorithm for this task. Although PROPAT can also propagate orbital position, we use SGP4 algorithm instead. Summarizing, the simulation process consists on (1) propagating orbital position (SGP4), (2) computing magnetic field for those positions (IGRF), (3) propagating attitude under B-dot control (PROPAT). Regarding to this control algorithm, we have test two different forms of operation: (1) applying continuous torque in each simulation step, and (2) generating intermittent torque, i.e, the cubesat is t_0 seconds unmanned, and t_1 seconds producing torque. As will be shown in VII-*B. Results*, this last method proves being more effective and less energy demanding.

A. Cubesat parameters

Table 1-3 contains the physical parameters of the cubesat and propagators used in the simulation.

Cubesat inertia matrix		
1/600 kg·m^2	0	0
0	1/600 kg·m^2	0
0	0	1/600 kg·m^2

Table 1: Inertia matrix corresponding to a 1 kg homogeneous 10 x 10 cm cube

Magnetorquers parameters			
Resistance	***Max Intensity***	***Turns***	***Area***
50 Ω	40 mA	40	0.0088 m^2

Table 2: Resistance, max. intensity, number of turns and area for each coil

Attitude and magnetic propagation		
Time	***Temporal step***	***Orbit modeled***
120 min	0.1 secs	ISS orbit

Table 3: Time, temporal step and type of orbit propagated

Must be highlighted that we have propagated ISS orbit (with TLE, or two-line element set [14], dating from 13th March), a station from which actual cubesats have been deployed.

B. Results

The simulation results prove that the magnetorquers are able to stabilize the satellite in less than two orbits, considering the initial angular speed given. *Fig.*14a shows that no component of angular speed is larger than 1 degree per second after the first orbit, and their values are basically zero after 1.5 orbits.

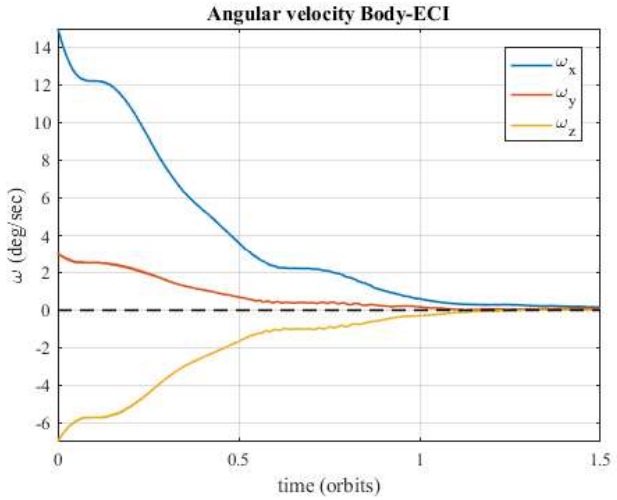


Fig.14a: Angular speed evolution in X, Y, Z axis

Energy consumption is shown in *Fig.*14b. It is clear than no more than 900 J are required for this task. This is understandable, even considering aerodynamical and gravity gradient torque that are simulated too. The first is very low, since 1U cubesat is a highly symmetrical tiny body.

Gravitational torque is even lower, due also to its symmetry and the fact that the satellite was considered a mass-homogeneous one.

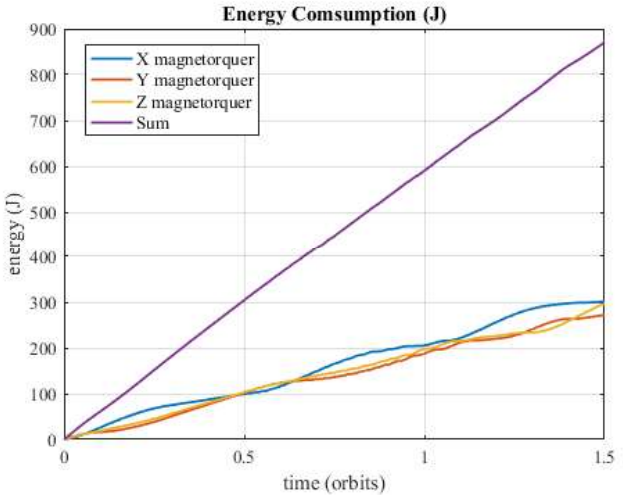


Fig.14b: Energy consumption evolution for each magnetor. and the sum of all

IX. STUDENTS APPROACH TO PROEJCT

The knowledge necessary to carry out this project should be presented to student in three sessions.

A. First session: theory

This section comprises physical and mathematical theory necessary to perform all simulations. Students should know how orbits are parametrized, and how IGRF model works. Attitude parametrization is also crucial, by the use of rotation matrices, Euler angles and quaternions. These geometrical elements play the same role concerning to reference frame transformation. In this project, only BODY and ECI frame are used. Contents are schematized in Fig.15.

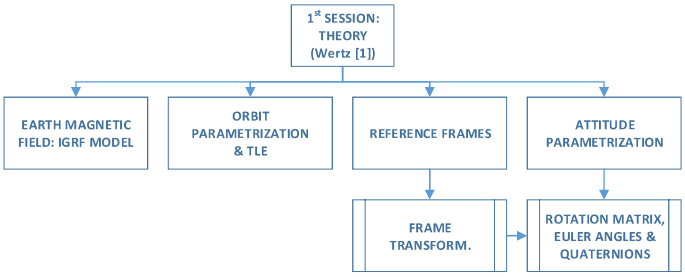


Fig.15: 1st Session contents

B. Second session: simulation

In this section, MATLAB software is given and explained to students: orbit, magnetic field and attitude propagation. B-dot algorithm should be implemented by students; they also could improve it by replacing basic derivative with a low pass filter [8] or using a more accurate estimation of the derivative. Contents are schematized in Fig.16.

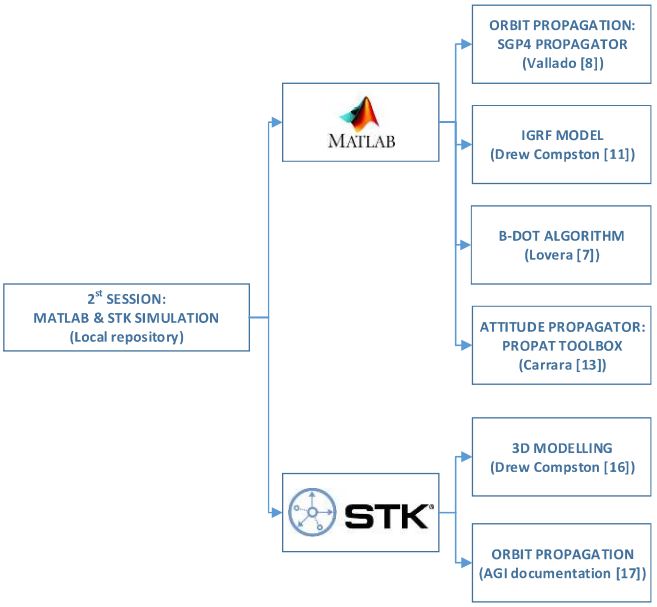


Fig.16: 2nd Session contents

C. Third session

In this section, C++ code required to be run in OBC is given. Furthermore, B-dot algorithm is implemented in the same way as it was in MATLAB. Contents are schematized in Fig.17.

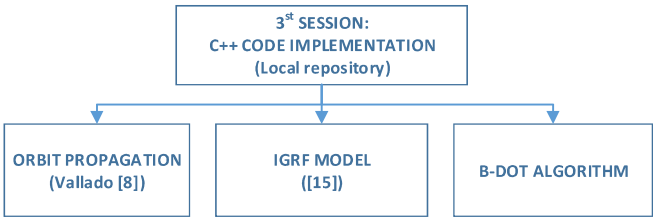


Fig.17: 3rd Session contents

X. STK FINAL TEST

B-dot algorithm is tested in STK environment and an orbital propagation video is generated.

Fig.18: STK orbital embedded video. Click to play.

XI. CONCLUSIONS

This research describes the design and construction of an economical and high-performance attitude control for 1U cubesat base on low cost 3D MEMS IMU sensor and Arduino CPU. Both STK and MATLAB scripts have been used to compare the propagated data and errors have been also modeled. With this work, we want to contribute with improvement of the engineering master's student skills in the field of aerospace physics knowledge.

ACKNOWLEDGMENT

The experiments and the testbed were supported by GranaSAT Aerospace in collaboration with the Telecommunication Master Student office. The authors also gratefully acknowledge the important contribution of lab technicians of the company DHT Technology.

This research received financial support from the national government of Spain, Project DEEPSAT RTC-2016-4644-3.

REFERENCES

[1] J. R. Wertz, (1978). Spacecraft Attitude Determination and Control

[2] W.J Larson, J.R. Wertz (1999). Space Mission Analysis and Design

[3] C. D. Hall, (2003). Spacecraft Attitude Dynamics and Control

[4] http://www.cubesat.org/about/

[5] Restrepo, Andres & Franco, Edinson & Pinedo, Carlos. (2014). Tri-axial Square Helmholtz Coils System to Generate Uniform Magnetic Field Volume

[6] https://www.agi.com/products/engineering-tools

[7] M. Lovera, "Magnetic satellite detumbling: The b-dot algorithm revisited," 2015 American Control Conference (ACC), Chicago, IL, 2015, pp. 1867-1872

[8] Vallado, David & Crawford, Paul. (2008). SGP4 Orbit Determination

[9] https://www.ngdc.noaa.gov/IAGA/vmod/igrf.html

[10] https://celestrak.com/software/vallado-sw.asp

[11] https://es.mathworks.com/matlabcentral/fileexchange/34388-international-geomagnetic-reference-field--igrf--model

[12] https://sites.google.com/site/sailboatinstruments1/proof

[13] Carrara, V. An Open Source Satellite Attitude and Orbit Simulator Toolbox for Matlab. DINAME 2015 – Proceedings of the XVII International Symposion on Dynamic Problems of Mechanics. Natal, RN, Brazil, Feb 22-27, 2015. (ISSN 2316-9567)

[14] https://www.celestrak.com/NORAD/elements/

[15] https://github.com/georgewfraser/drawing/blob/master/geo/IGRF/geomag60.c

[16] https://www.dropbox.com/s/d9kqnpgrt4vvdfu/CAD-Blender-Collada-STK.zip

[17] http://help.agi.com/stk/

[18] http://www.nuts.cubesat.no/upload/2013/03/01/masteroppgave-gaute-brathen.pdf Section 2.6.1