TALLINN UNIVERSITY OF TECHNOLOGY
Faculty of Information Technology
Department of Computer Engineering

IDK40LT

Peeter Org 123513

SOFTWARE ARCHITECTURE AND DEVELOPMENT TOOLS API FOR ATTITUDE CONTROL SYSTEM OF TUT NANOSATELLITE

Bachelor's thesis

Supervisor: Martin Rebane
MSc
Lecturer

Tallinn 2016

TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutitehnika instituut

IDK40LT

Peeter Org 123513

TTÜ NANOSATELLIIDI ASENDIKONTROLLISÜSTEEMI TARKVARA ARHITEKTUUR JA ARENDUSVAHENDITE API

Bakalaureusetöö

Juhendaja:	Martin Rebane
	MSc
	Lektor

Tallinn 2016

Author’s declaration of originality

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Author: Peeter Org

23.5.2016

Abstract

Main subject of this thesis is higher level software architecture of TUT nanosatellite’s attitude determination and control system. Secondary subject is to develop a software application programming interface for further development of software and simulation environment of this system.

The architecture is developed based on attitude determination and control system’s hardware design, gathered software requirements and experience from previous student nanosatellite missions. Software is written in C, based on hardware design of development board being used

This thesis is written in English and is 38 pages long, including 5 chapters and 9 figures.

Annotatsioon
TTÜ NANOSATELLIIDI ASENDIKONTROLLISÜSTEEMI TARKVARA ARHITEKTUUR JA ARENDUSVAHENDITE API

Selle töö peamine eesmärk on välja töötada tarkvara arhitektuur TTÜ nanosatelliidi asendikontrollsüsteemi jaoks. Teisejärguline eesmärk on välja töötada rakendusliides edasiseks arendustegevuseks.

Tarkvara arhitektuur luuakse selle süsteemi riistvara disaini, kogutud nõuetele tarkvarale ja eelmiste tudengisatelliidi missioonide põhjal. Tarkvara arendatakse programmerimiskeeles C, kasutusel oleva arendusplaadi riistvarast lähtudes.

Lõputöö on kirjutatud inglise keeles ning sisaldab teksti 38 leheküljel, 5 peatükki ja 9 joonist.

List of abbreviations and terms

ADCS	Attitude determination control system
API	Application programming interface
BOM	Beginning of message
DPS	Degrees per second
ECEF	Earth-centered, Earth-fixed reference frame
ECI	Earth centered inertial reference frame
EOM	End of message
GS	Ground station
IGRF	International geomagnetic reference frame
NORAD	North American Aerospace Defense Command
OBC	On-board computer
OS	Operating system
PC	Personal computer
PCB	Printed circuit board
RXNE	Receive register not empty
SBRF	Satellite body reference frame
TLE	Two line element set
TUT	Tallinn University of Technology
TXE	Transfer register empty
UART	Universal asynchronous receiver/transmitter
USART	Universal synchronous asynchronous receiver/transmitter

Table of Contents

List of figures

1 Introduction

1.1 About TUT nanosatellite project

1.1.1 Project

Goal of TUT nanosatellite project is to provide students with theoretical and practical experiences of building a satellite. [1]

Scientific/technological goal of this project is to test line-scan camera on a nanosatellite. Line-scan camera has not yet been installed on a nanosatellite.

1.1.2 CubeSat standard

TUT satellite will be built according to CubeSat standard. Main characteristics of a CubeSat are that it's cube-shaped, with 10 cm sides and its weight is below 1.33 kg. This standard was developed in 1999 by California Polytechnic University and Stanford University with purpose to provide access to space for small objects [2]. Those satellites are launched as secondary payloads on launch vehicles or put in orbit by deployers on the International Space Station. TUT satellite will be launched as a secondary payload on a larger satellite.

1.1.3 Subsystems

There will be following subsystems on board TUT satellite:

Mechanical structures - tasked with structural integrity of the satellite and deployment of solar panels.

Communications system - tasked with communication between satellite and ground station (GS).

Power system - tasked with power generation, power distribution between subsystems, battery charging and emergency communications.

On-board computer (OBC) - tasked with primary computing tasks and control of the satellite.

Attitude determination and control system (ADCS) - tasked with determining and controlling satellite’s attitude (orientation) in space.

Optical payload - tasked with capturing images using line-scan camera.

1.2 Goals of this work

Primary goal of this thesis is to develop architecture and high-level application programming interface (API) for TUT nanosatellite’s ADCS. In order to achieve this, it is necessary to understand the working principles of each component of ADCS, to figure out what is the input and output value of each software component, what the sequence execution is and which components can work concurrently.

Secondary goal is to develop application programming interface (API) for connecting simulation tools with ADCS development board (PCB). Code for this API is written in C.

1.3 ADCS

1.3.1 Task

ADCS, which stands for attitude determination and control system, is a satellite’s subsystem tasked with determining and controlling satellite’s attitude (orientation) in space. This system usually has its own processor, sensors and actuators independent of the rest of the satellite’s subsystems. There are various different methods being used for both determining and controlling satellite’s attitude, most common of which are listed below.

1.3.2 Sensors

Various different sensors can be used to determine satellite’s attitude, more common methods include:

Sun sensors - Sun sensors can tell, by intensity of light falling on photo sensors, the direction of Sun in satellite’s body reference frame (SBRF). By comparing sensor result with an on-board Solar system model, satellite’s attitude can be determined. There should be a sun sensor on each side of the satellite to ensure that Sun’s direction can be tracked from all directions. Sun sensors are reliable, accurate, simple and lightweight, but they cannot be used during eclipse [3].

Gyroscopes - gyroscope is a spinning wheel, which retains the orientation of their spinning axis due to conservation of angular momentum. Modern gyroscopes are lightweight and measure angular velocity and acceleration accurately, but due to axial precession and fact that gyroscope doesn’t keep its orientation when it’s not spinning, gyroscopes alone cannot be used to determine satellite’s attitude.

Magnetometers - magnetometers measure magnetic field and its strength in 3 dimensions. Comparing magnetic field reading to an on-board Earth’s magnetic field model can be used to determine satellite’s attitude. Magnetometers are simple lightweight devices that provide relatively good accuracy with respect to their mass and size [3].

Star trackers - star tracker is a camera mounted on board the satellite that takes pictures of stars and compares it to an on-board model of night sky to determine attitude. Because stars are very small and their position in sky changes very little, star trackers are very accurate devices for determining attitude. They are, however, expensive and bulky, because a baffle is needed to block stray light from Sun, Earth and Moon. Furthermore, star trackers need relatively large computational power and they need the satellite to be still, which means star tracker is usually not a very good option for nanosatellites.

Horizon sensors - horizon sensors work the same way as Sun sensors, but they have a much narrower field of view, thus they can only tell if Earth is visible or not. Horizon sensors can, at the right angle, detect Earth’s horizon and thus two angles of attitude can be determined.

1.3.3 Actuators

ADCS can use following devices to control attitude:

Thrusters - thrusters use propellant to induce force on satellite’s body. Thrusters are, compared to other methods, fast and effective, but heavy and take up significant amounts of space in satellite’s body. Because the amount of propellant that can be taken on board is limited, thrusters can only be used for certain amount of time. Having propellant on board also causes thermal and configurations problems [3]. Because of this, thruster systems are usually not a favored option aboard nanosatellites.

Gravity booms - gravity boom is a passive method that uses only satellite body's mass distribution and Earth's gravitational field to control attitude. Because Earth's gravity decreases with height, a tidal force can be generated by extending a weight perpendicular to the orbit. The advantages of this system are low energy consumption and no propellant needed, but it is slow and adds considerable weight to satellite.

Inertial wheels - idea of inertial wheels is to accelerate a body inside the satellite to create an opposite force to satellite's body. This system is fast and relatively accurate and doesn't need propellants, but is heavy, takes up considerable space inside satellite's body and having moving parts on board may cause reliability issues.

Magnetorquers - magnetorquer method uses magnetic coils, which generate a magnetic field. This magnetic field's interaction with Earth's magnetic field generates torque on satellite's body. Magnetorquers are very reliable because they contain no moving parts, relatively lightweight and need no propellant. Magnetic coils are a very popular means of controlling attitude of nanosatellites, many student satellite missions like SwissCube [3], EstCube [4] and AAUSAT [5] have used magnetorquers as actuators for their ADCS system.

2 Design of ADCS

2.1 Hardware design

TUT nanosatellite will be using magnetorquers method for attitude control. For attitude determination the ADCS will use a combination of Sun sensors, magnetometers and gyroscopes.

Because losing either magnetometer or gyroscope would result in a mission failure, both of these sensors are duplicated for redundancy purposes. Magnetorquers are controlled by using digital-analog converters on board the system. For communication between subsystems, the satellite will have an RS485 bus. TUT satellite's ADCS will use STM32F MCU, which is based on ARM Cortex M4 architecture [6].

ADCS's hardware schematic is shown in Figure 1.

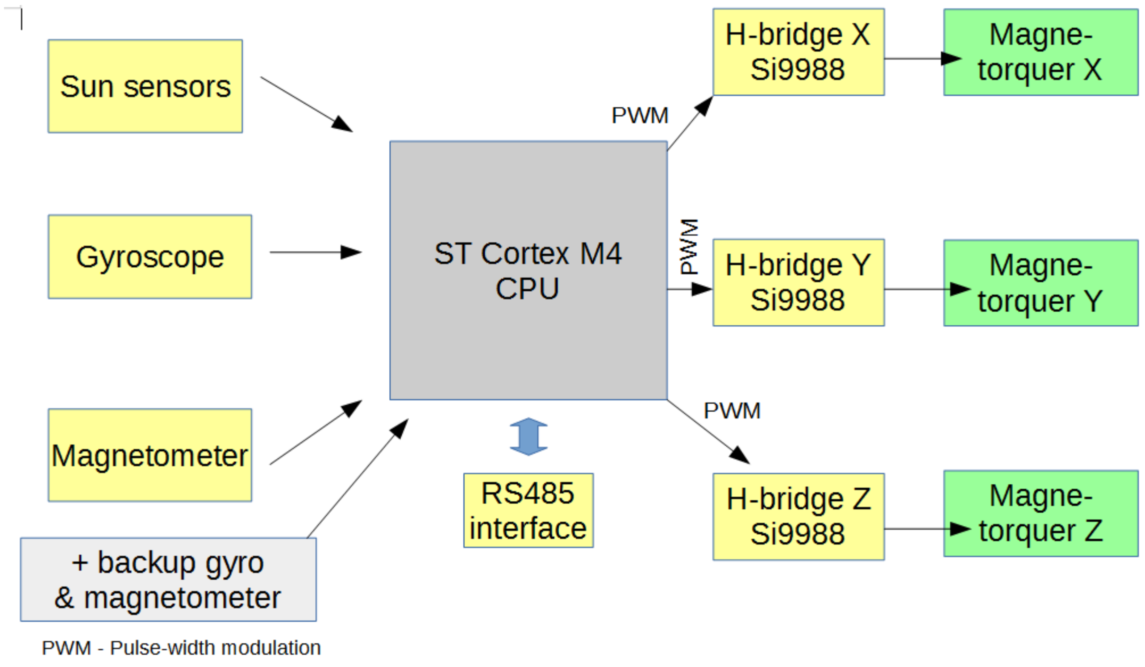

Figure 1. ADCS hardware schematic [16]

2.2 Software design

2.2.1 Overview

The end goal of ADCS's software is to calculate currents for magnetorquers. This is done according to satellite's current attitude, satellite's desired attitude and Earth's magnetic field's state. Before currents can be calculated, attitude needs to be determined. To determine attitude, satellite's location in Earth-centered, Earth-fixed inertial reference frame (ECEF) is required. ECEF is a Cartesian coordinate system where point [0, 0, 0] is defined as Earth's center of mass, hence Earth-centered, and its axes are aligned with International Reference Pole and International Reference meridian, hence Earth-fixed [7]. Satellite's position can be calculated from time and a model of the orbit, which, by popular convention, is represented in a two line element (TLE). TLE along with current time will be uploaded from GS via radio channel. When satellite's location is determined, attitude can be calculated by using sensor data. Attitude determination is done by measuring magnetometer and Sun sensors and comparing those read results with on-board models of Earth's magnetic field and Sun's direction. Once attitude has been determined, it can be controlled by applying currents to magnetorquers which will generate torque on satellite's body by interacting with Earth's magnetic field.

Scheme for controlling attitude is shown in Figure 2. This schematic, drawn based on known components and requirements for this system, forms the basis for work. Rectangular shapes represent software components, circular shapes represent hardware components.

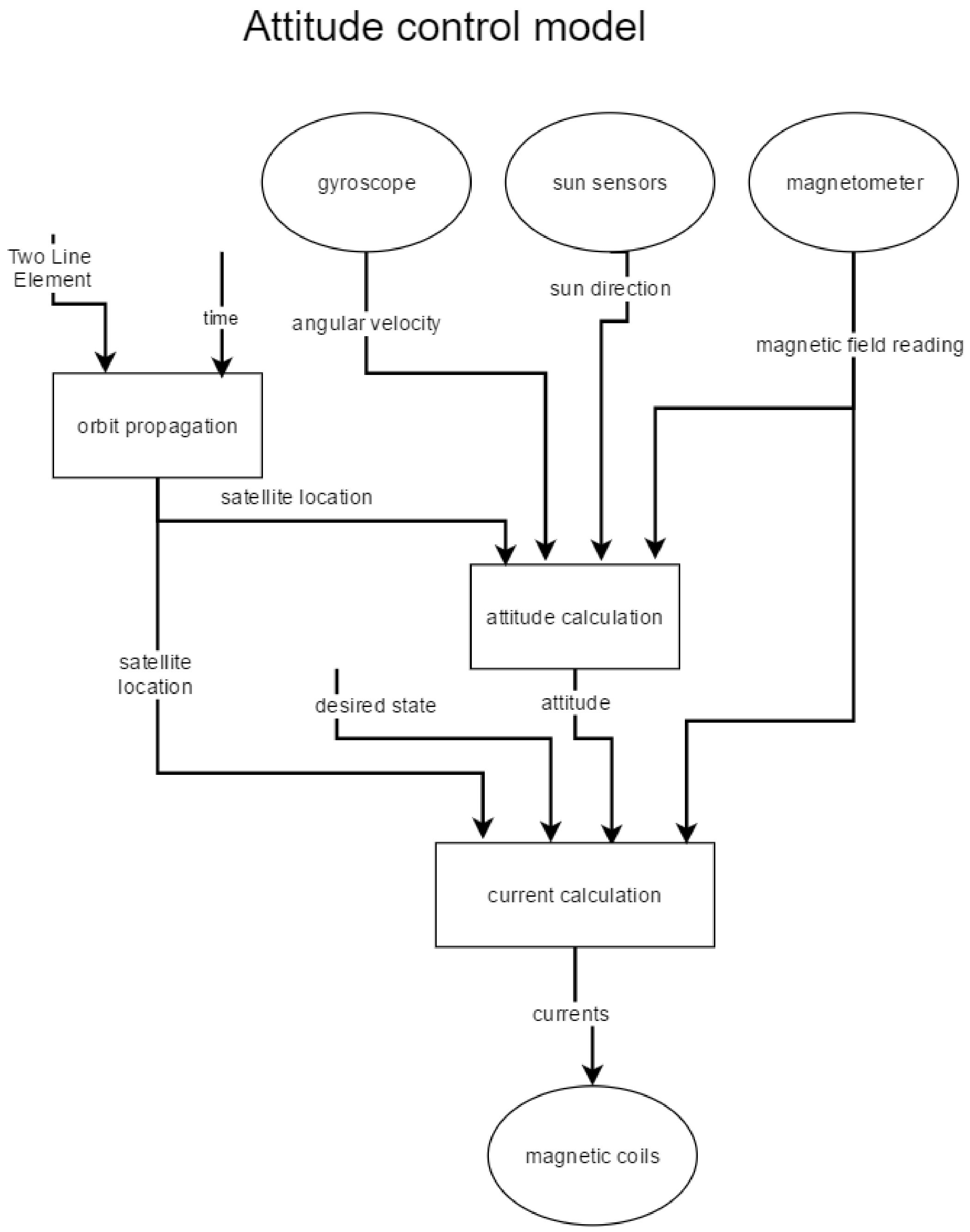

Figure 2. Attitude control model. Author’s drawing.

2.2.2 Requirements for attitude control software

Based on literature and project’s work group meetings, following requirements for software of TUT satellite’s ADCS have been gathered author:

1. Because the magnetic field generated by coils interferes with magnetometer reading, applying torque and reading from magnetometer cannot be done at the same time, therefore attitude control has to be done in a cycle
2. Torque cycle should be reasonably short, (EstCube-1's attitude control cycle was 3 Hz [4], a similar frequency will be estimated for TUT nanosatellite in this work), because if it takes too long time to calculate the currents, satellite might make a full turn and in this case using magnetorquers to control attitude would be impossible
3. To fulfil all tasks required from ADCS, the system has to work in several different working modes, which will be specified later in this work.
4. The system has to be able to communicate with other subsystems at all times, regardless of working mode or current stage in working cycle.
5. Because latch-ups and bit-flips may occur in space due to radiation [8], the system has to be capable of tolerating hard resets in any given time, and the system has to be able to recover from a hard reset.
6. Attitude should be determined and controlled with accuracy of at least 2 degrees [9], required to accomplish the satellite's main goal, which is taking pictures of Earth.

2.2.3 Working modes

Due to mission requirements and hardware design, several different working modes are required:

1) **Detumbling/stabilizing** – when the satellite is lauched, it will probably be spinning uncontrollably, so the first thing that needs to be done after lauch is to stabilize the satellite to make it possible to communicate with ground station.
2) **Charging** – because TUT nanosatellite will be using deployable solar panels, for maxiumum charging efficiency these panels should be pointed towards the Sun
3) **Taking picture** – satellite's camera needs to be pointed towards a certain location on Earth and kept as sable as possible to take a picture.
4) **Cooling** – when a side of satellite gets too hot due to solar radiation, it should be put in a controlled spin to divide heat between sides.
5) **Communication** – antennas need to be turned towards the GS to ensure maximal communication speed. Because the satellite moves in orbit it's orientation relative to GS changes constanly, which means satellite's body needs to be put in a controlled spin to keep antennas pointed towards GS.

6) **Idle** – when no activity is required from ADCS, this mode should be switched on. To save battery, most of systems should be powered down, for example gyroscope. Some tasks, like orbit propagation, should still be performed.

2.2.4 Main components

Components of the software can be divided into larger components as following:

Earth's magnetic field model – Due to geographical variations in Earth's magnetic field, the field's strength and direction cannot be predicted by satellite's location alone. This means a separate model for Earth's magnetic field on board the satellite is required. Because attitude will be determined by comparing magnetometer reading to this model's estimation, this model needs to be as accurate as possible. Input of this component is satellite's location in ECEF. Outputs of this component are magnetic field's strength and direction in ECEF.

Orbit propagator – Knowing satellite's location is essential for both determining and controlling satellite's attitude. When orbital elements of the satellite are known, a suitable prediction formula can be used to estimate the satellite's location at any given time. Orbital elements will be provided in the form of TLE by NORAD as a free service, which will be uploaded to satellite from GS. Inputs for this task are Two Line Element and real time, outputs are satellite's location in ECI and ECEF.

Sun vector model – sun sensor readings needs to be compared to an on-board model of solar system to assist in determining satellite location. Inputs for this component are real time and satellite's location in ECEF. Output of this component is sun's direction in ECEF.

Attitude determination algorithm – because attitude and current calculation takes time, and effectiveness and accuracy of the magnetic coils relies on attitude for which the currents were calculated, attitude should be predicted ahead for current calculation algorithm. Inputs of this component are therefore attitude estimated by Sub vector and Earth's magnetic field model, angular velocity from gyroscope and time (how much ahead should attitude be predicted). Because sun is not visible at all times, the algorithm needs to perform its tasks with data from magnetic field model and gyroscopes only.

Earth's albedo model – Earth's albedo (light reflected from Earth's surface and atmosphere) is going to interfere with sun sensor's reading, so the reading has to be compared to an on-board model of Earth's albedo to compensate for this.

Perturbation model – gravitational forces from Earth, Moon and possibly other bodies and also resistance from atmosphere and other sources are going to change the orbit of the satellite, which may cause erroneous predictions of satellite's location, which would cause errors in attitude estimation. To avoid that, an on-board model of possible perturbations is necessary to take this into account.

B-dot algorithm – after launch, the satellite is going to be in an uncontrolled spin. In this situation, communications cannot be established with GS. This means that the orbital elements cannot be uploaded to the satellite, so satellite's location cannot be calculated. Without satellite's location, most of software components cannot be used. To establish communication with ground station, the satellite needs to be brought to a condition of sufficiently small angular momentum. This process is called detumbling, and for this task a separate algorithm is needed. One of the most common methods for this is so-called b-dot control law which will probably be used in TUT satellite. Component schematic of ADCS in detumbling mode is shown in Figure 3.

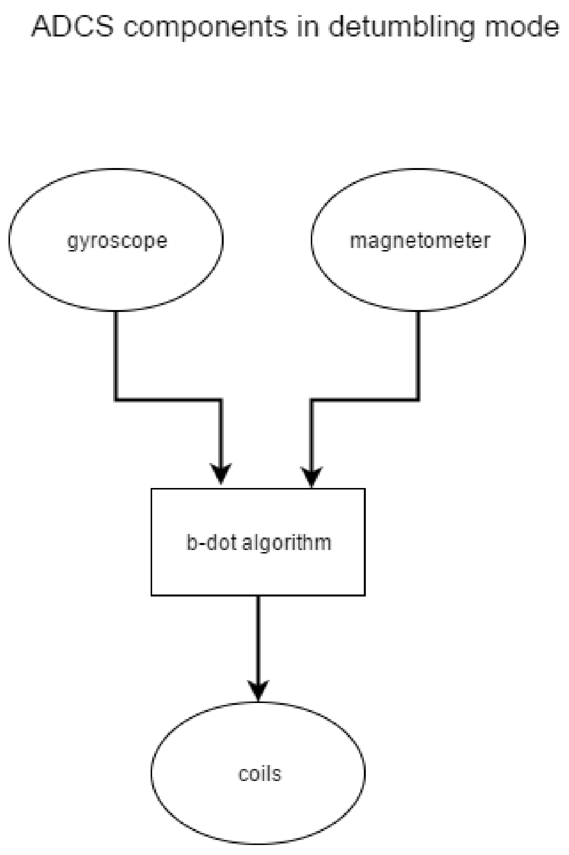

Figure 3. ADCS components in detumbling mode. Author's drawing.

Current calculation algorithm – once all necessary information has been obtained, current can be calculated for the magnetic coils. Input values for this component are going to be desired state of satellite body, satellite's current state (attitude and angular velocity) and magnetometer reading. Output values are going to be currents for the three magnetic coils. Current must be calculated for different desired states. For example, in taking picture mode, desired state is certain attitude. For cooling the satellite, desired state is a certain angular velocity. Because of this, there is no certain format for desired state. It may be certain attitude, certain angular velocity or a combination of both. For this reason, to reduce complexity of the software, a separate controller should be made to handle transitions between the states, and a current calculator which calculates the actual currents. The controller passes current calculator an angular acceleration vector, according to which the current calculator calculates currents to generate angular acceleration in desired direction. Current calculation scheme is shown in Figure 4.

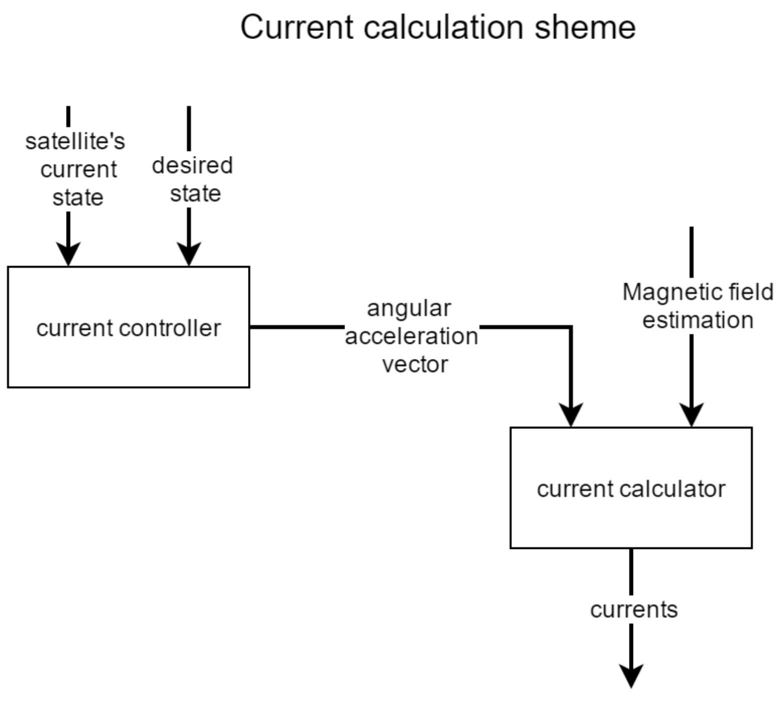
Current calculation sheme
satellite's current state
desired state
current controller
angular acceleration vector
Magnetic field estimation
current calculator
currents

Figure 4. Current calculation scheme. Author's drawing.

Controller – because sensors that are required to determine and control attitude are on board ADCS, only general commands, like „make satellite spin at given speed“ or „point solar panels towards Sun“, will be received from on-board computer. The actual numeric values of the desired state, like satellite's attitude in ECEF or angular velocity should be determined by

ADCS. To do this, a separate software component should be used. This component will be using all the data available on board ADCS, and perhaps on other subsystems, to calculate desired state and pass it to current calculation algorithm.

Communication between components in normal working mode is shown in Figure 5. Rectangular shapes represent software components and circular shapes represent hardware components.

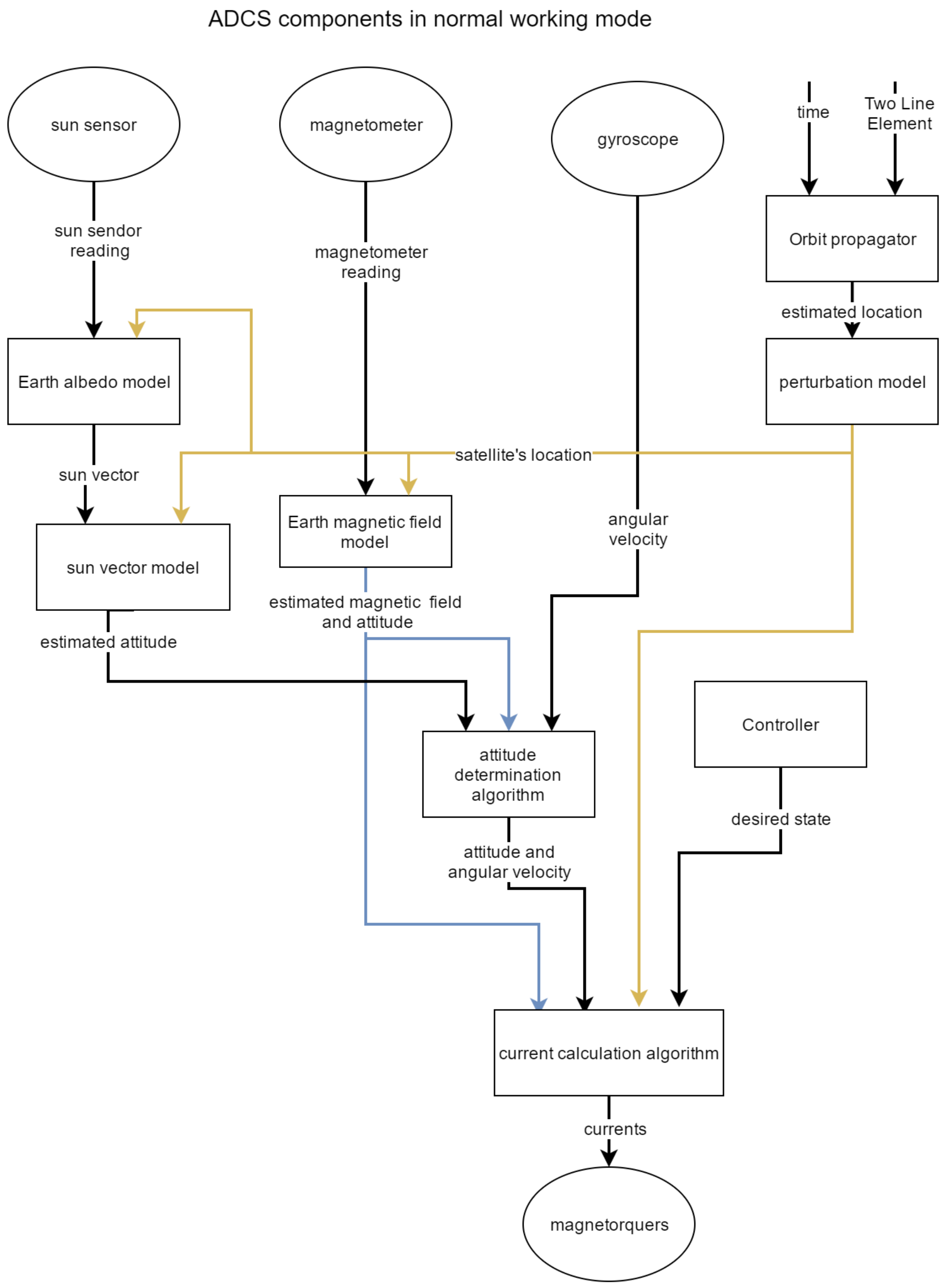

Figure 5. ADCS components in normal working mode. Author’s drawing.

2.3 Operating system

TUT satellite's ADCS will running on FreeRTOS, which is an open-source cross platform real time operating system [10]. The advantages of this OS is that it is completely free, open-source, lightweight and easy to use. Many previous student satellite missions have used FreeRTOS.

2.4 Simulation

A simulation environment is needed to develop and test the software. Modelling and simulation will be done mainly in Matlab and Simulink. The goal of simulation environment is to estimate the effect of currents applied by ADCS on satellite's attitude and generate input data for the ADCS accordingly. Scheme of ADCS's simulation environment is shown in Figure 6.

Simulation software should contain following components, some of which can be reused from those on board the ADCS:

ADCS – this is the component that will be simulated, it will be running on a test board, its inputs are going to be data from sensors and outputs are going to be currents for the magnetic coils. It will be connected to a PC with simulation environment through a serial port.

Sensor emulators – emulators for Sun, magnetic field and angular velocity sensors are needed. Main goal of these emulators is to add noise to sensor readings to create space-like conditions for ADCS. Inputs for those components are therefore actual conditions that the sensors will be measuring and output will be what sensor's reading would be like based on that.

Coil model – emulator for magnetic coils is needed. Impedances of coils, satellite body's influence on magnetic field and coil's actual generated magnetic field needs to be emulated. Input for this component will be values written to digital-analog converters and output will be magnetic field generated by the coils.

Earth's magnetic field model – goal of this component is to generate input data for magnetometer emulator and torque calculation components. While Earth's magnetic field model on board the satellite returns magnetic field's vector in ECEF, this model should return magnetic field’s vector in SBRF, which is why satellite’s attitude is also required as input along with satellite’s position.

Orbit model – this model is going to calculate satellite’s location in this simulation. This will work the same way as orbit propagator on board the system.

Sun vector model – to create input for Sun sensor emulator, a component is needed that calculates sun’s assumed direction based on time, satellite’s location and attitude.

Torque calculation – to calculate torque created by interaction between coil’s magnetic field and Earth’s magnetic field this component is needed. Inputs are Earth’s and coil’s magnetic fields vectors and output will be torque vector.

Satellite body model – to calculate effect of torque on satellite’s body, a component that includes model of satellite’s moment of inertia is needed. This component takes torque vector as input and outputs angular velocity.

Attitude calculation – this component remembers satellite’s current attitude and updates this based on angular velocity of satellite’s body. Output will be satellite’s attitude in ECEF.

Clock – for synchronization between components and because satellite’s clock and actual time may not be in sync, the simulation environment should have its own clock.

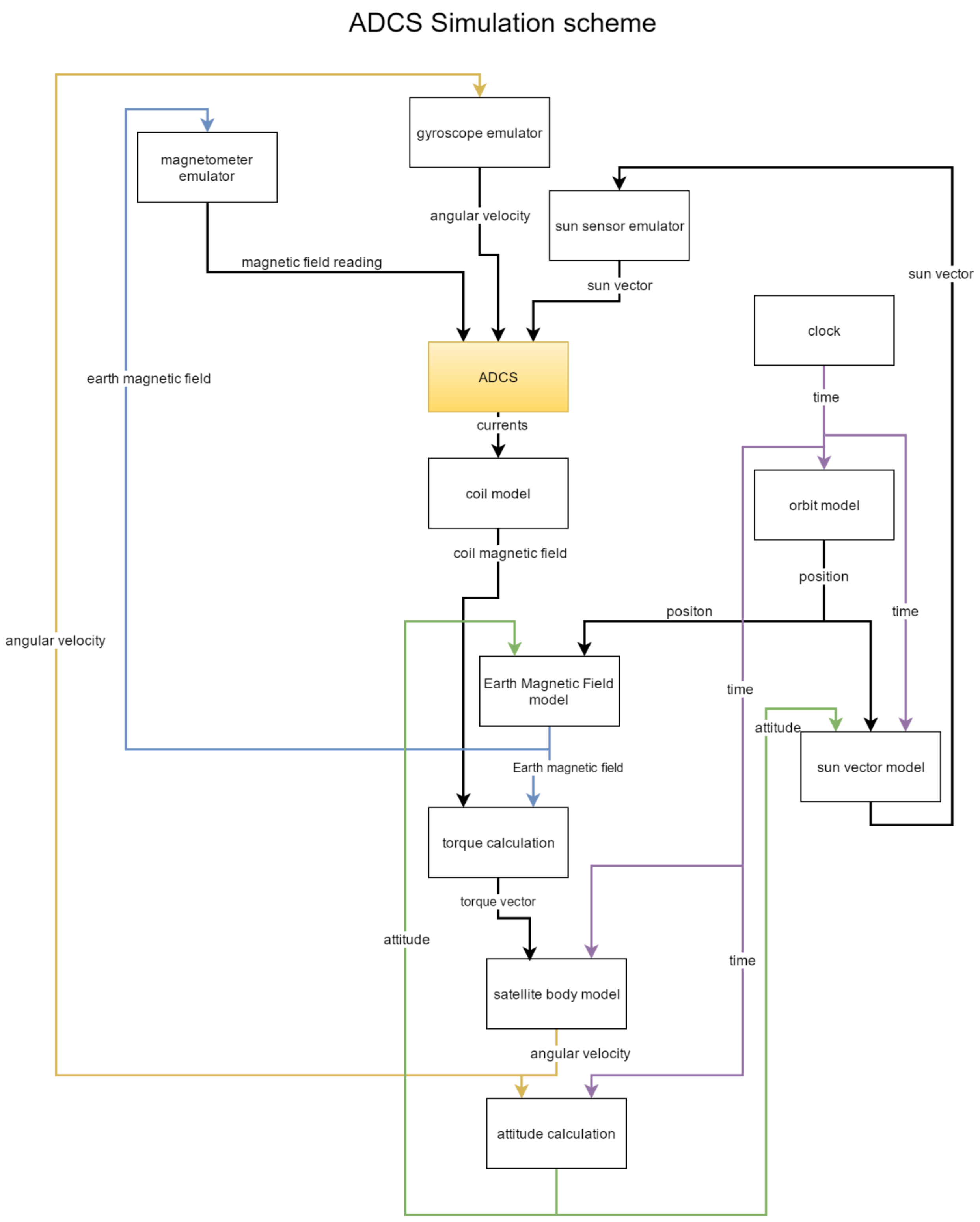

Figure 6. ADCS simulation scheme. Author’s drawing.

3 Tasks within operating system

3.1 Task allocation

The amount of different tasks should be minimal to increase efficiency of the software, but anything that needs to be calculated in parallel has to be in separate task. For those reasons, tasks within the operating system will be allocated as following

1. **Bus communication** – Due to requirement 4, a task should be always listening to signals and data from bus and sending data out when needed. This task contains communication protocols for the bus and send-receve buffers for data. Whenever there's data to be sent and bus is open, this task should be activated with highest priority until messages and data is sent.
2. **Attitude and current calculation** – calculating attitude and current is going to be done in a sequence, which means this can be done within the same task. This task is computationally most consuming and also most complex, because all the working modes realised withing this task.
3. **Orbit propagation** – satellite's location is something that needs to be periodically updated and predicted, so this should be performed in a separate task. This task is going to be woken up after fixed time intervals, regardless of working mode, and calculates a list of locations where satellite is going to be within upcoming short period of time, and stays suspended until this period is over, to be woken up to update locations again.
4. **Coil control** – a task needs to apply and maintain currents of the coils during torque cycle, and therefore should be a separate task to do this. When currents are calculated and ready to be applied, this task is woken up, currents are applied and task is put back to sleep. For a certain amount of time, this task is periodically woken up to make adjustments to currents and when this time is over, this task should is put in a suspended mode until woken up again by attitude and current calculation task.
5. **Sensor reading** – one task should be concentrated solely on reading data from sensors, incase some other subsystem requires sensor data. This task should read data from all three different sensors, but because magnetometers are not useful when magnetorquers are

active, this task should have two different working modes, depending on current phase in torque cycle. When magnetorquers are inactive, magnetometer reading can be used and then magnetometers should be read, otherwise magnetometers should be ignored to save processing time. Because sun sensors are useless during eclipse, this task should have another two working modes, independent of whe first two. When the satellite is in eclipse, sun sensor readings should be ignored.

Communication between tasks is shown in Figure 7.

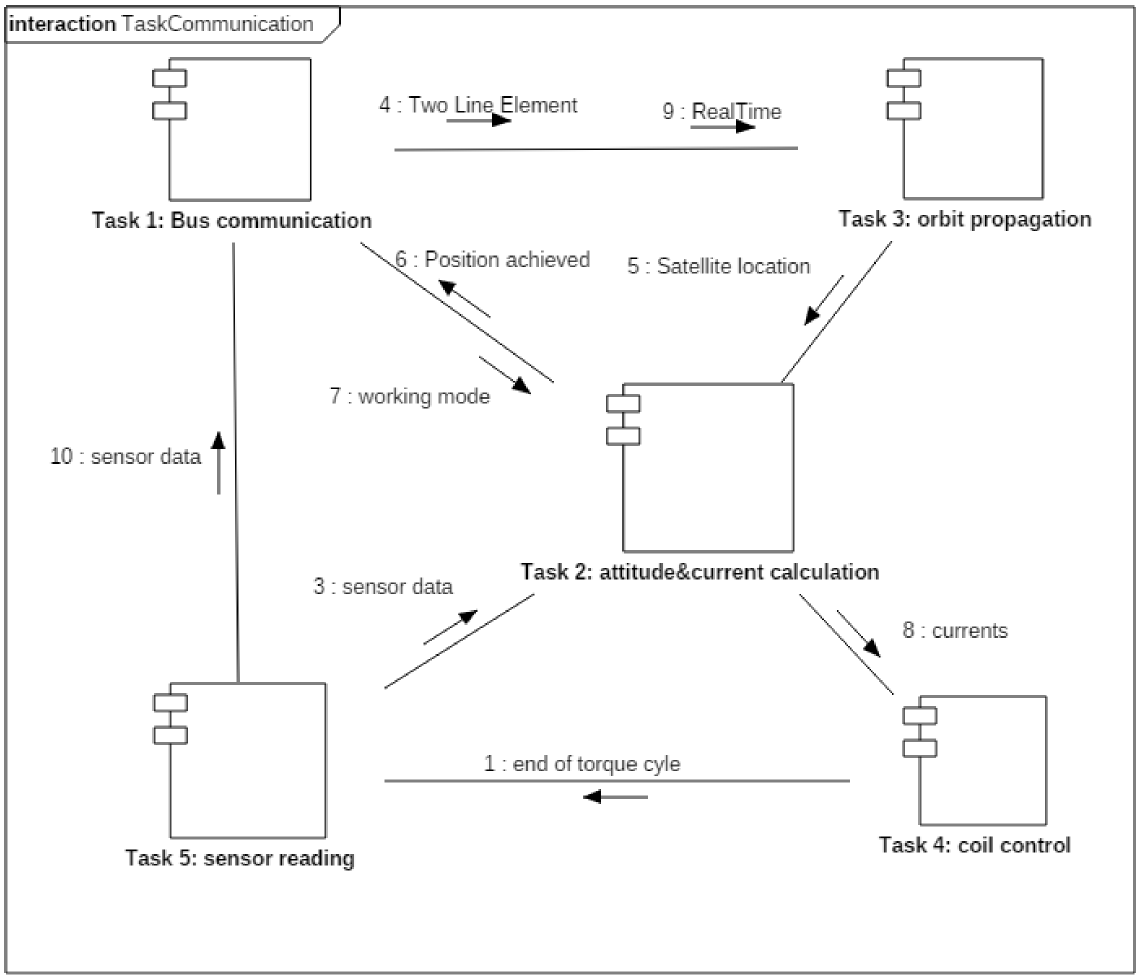

Figure 7. Communication between tasks. Author’s drawing.

3.2 Components within tasks

Components should be included within OS's tasks as following

Attitude and current calculation task should include following components:

Attitude determination algorithm has a central role in this task, along with **current calculation algorithm.** These components will be initiated in a sequence because current calculation algorithm is heavily dependent on attitude determination algorithm.

Controller will also be included within this task because current calculation algorithm needs the desired state as input.

Sun vector and Earth's magnetic field models hold a key role in determining satellite's attitude, which means they should be included within attitude and current calculation task.

B-dot algorithm- some other components that might be required to control coils are going to be contained within this task, because of this the B-dot algorithm should also be included within this task.

A smaller component is required for measuring time it takes to calculate attitude and current, to be used as input for attitude determination algorithm.

Orbit propagation task's task is centered on the **orbit propagator**. Other systems and tasks are only going to need satellite's final positon. To calculate satellite's final position, **perturbation model** should also be included within this task.

Sensor reading task should include:

Earth albedo model- because other tasks and systems are only going to need Sun's direction and not sun sensor reading itself, Earth's albedo should be filtered out within this task before sending data to other systems.

Additional smaller components, like filters for sensors are also included sensor reading task.

3.3 Torque cycle

As stated in requirement 1, attitude control needs to work in cycles. Cycle should consist of at least three different stages where:

- magnetorquers are idle and Earth’s magnetic field can be accurately measured
- magnetorquers are active to apply torque to satellite’s body
- magnetorquers are cooling down after applying torque, which takes time due to magnetic inductance

Two types of cycles have can used to control satellite’s attitude, first of which consists of four stages: measurements, calculation, torque and cooldown. The advantage of this cycle is that calculations are made with most up-to-date data and torque is applied right after calculations are finished, so there would be little possibility of any change happening with satellite’s attitude between calculations and torque, which gives this configuration good accuracy.

Second type has three stages: measurements, torque&calculations and cooldown. Applying and maintaining currents on magnetorquers requires little to no processing power, but will be maintained for a noticeable amount of time. During this period, currents for the next torque cycle can be calculated, shortening overall torque cycle by the amount of time it would be required to calculate currents and therefore significantly increasing the speed at which satellite’s attitude can be changed. Trade-off of this cycle is accuracy, because currents would be calculated for the next cycle, by time of which the satellite’s angular velocities will have changed due to torques applied on current cycle. This change, however, is probably insignificant because the torques created by magnetic coils are very small. Another trade-off is slightly more complex software because satellite’s attitude would need to be predicted further ahead and additional variables would be required to store values for both current and next torque cycles.

If accuracy loss of the second cycle is small enough, the second cycle should be used. This means a rough estimation of how much attitude can be changed within one cycle is required. The estimation will be done according to worst-case scenario, which in this case is the biggest possible change to satellite’s attitude during one cycle. For attitude control overall this would be the best-case, but for second torque cycle this is the worst case. Magnetorquers ability to apply torque to satellite's body is characterized as magnetic moment **M**, whose units are **Am^2** (Ampere per meter squared) [11].

The aim for TUT satellite is to get a magnetic moment of 0.12 Am^2 from the coils [9]. The torque produced by magnetorquers is cross product between Earth's magnetic field strength **B** in teslas and coil's magnetic moment [11]:

$$\mathrm{T} = B \times M$$

Earth's magnetic field in its strongest point is 65 microteslas. [12] Cross product is largest when the two vectors are perpendicular (which would be the worst-case), in which case length of the resulting vector is equal to product of the lengths of those two vectors:

$$|\mathrm{T}| = |B| \cdot |M| = 0.12Am^2 \cdot 65\mu T = \ 7.8\ \mu N$$

TUT nanosatellite will be using deployable solar panels, which significantly change and increase its moment of inertia, but in this calculation, for simplicity reasons and the fact that smallest moment of inertia is worst case, equation for a cube will be used. Moment of inertia of a solid cube is following [13]:

$$\mathrm{I} = \frac{ms^2}{6}$$

Where **m** is mass and **s** is length of a side. TUT satellite's mass will be roughly 1.2 kg and length of side, as set by CubeSat standard, 0.1 meters. This means the satellite's moment of inertia will be at least:

$$\mathrm{I} = \frac{ms^2}{6} = \frac{1.2 \cdot 0.1^2}{6} = 0.002\ kg \cdot m^2$$

Equation to calculate angular acceleration is:

$$\alpha = \frac{\mathrm{T}}{\mathrm{I}} = \frac{7.8\mu N}{0.002kg \cdot m^2} = \ 3.9 \cdot 10^{-3}\, rad/s^2 \approx 0.223\, deg/s^2$$

This means that even if torque phase in the cycle would last 1s (which would be a very long time for a torque phase), change in angular speed would be only $0.223\, deg/s$, which means even if torque cycle would last a couple of seconds, the changes that would happen during that cycle would not surpass accuracy requirement of 2 degrees. This means that the second type of torque cycle should be used.

Activity of tasks within the cycle is shown in Figure 8 (this figure does not represent the relative durations of stages). Bus communication and orbit propagation tasks are running in background during all phases of the cycle, attitude and current calculation task will be woken up at the beginning of calculations & torque phase, coil control is woken up at the beginning of calculations & torque stage as well, but running in background as only very little calculations are required for maintaining currents. Sensor reading&filtering task will be running as primary task during sensor measurements phase, because sensor measurements need to be read multiple times in a quick succession to get an accurate reading. During other stages, reading & filtering task will be running in background to update readings continuously.

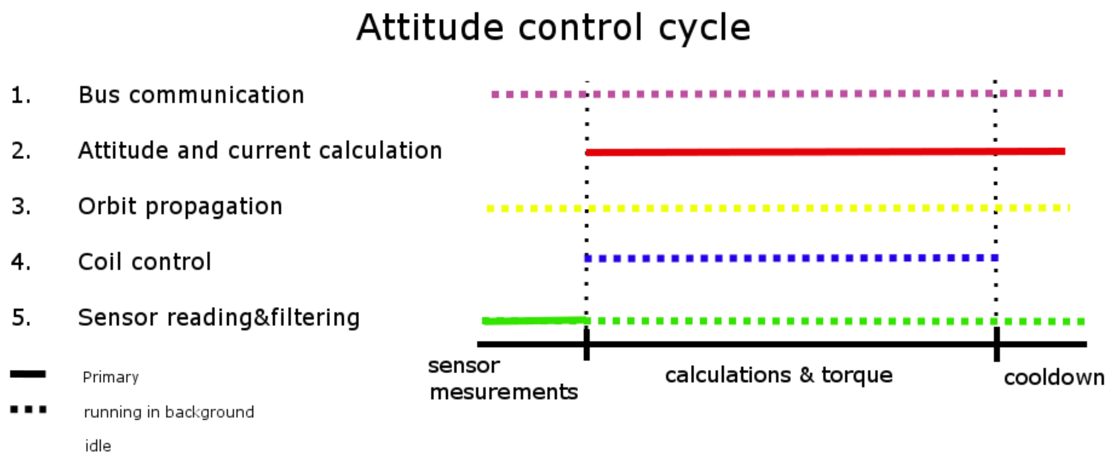

Figure 8. Attitude control cycle. Author's drawing.

4 Developed software

In following chapter the secondary goal of this work, which is development of software API for simulation and further software development, will be covered. Two different APIs are made, first is API for establishing communication between test board and PC with simulation environment and second is API for configuring and gyroscope on the test board, so actual gyroscope data can be used in simulations.

4.1 Development environment

Software is run and tested on STM32F3 Discovery development board, which is connected to computer through USB cable. All code for this project is written in C.

Development is done in Eclipse IDE, which has Bleeding Edge Toolchain and GDB debugger installed on it. Communication with test board is established through OpenOCD software. Source code is compiled for ARM Cortex M4 architecture using Bleeding Edge toolchain, GDB connects to OpenOCD through localhost for downloading code and debugging.

4.2 Com port API

Before any further development can be done, communication has to be made possible between PC and the development board. This will be done by using Prolific PL2303 USB to UART Bridge, which is connected to USART3 on development board side and to USB connector on computer side. This device opens a COM port inside the PC which can be accessed by a variety different programs, like Matlab.

4.2.1 Design

A separate OS task is required for sending and receiving data. To save processor’s resources, interrupts are be used for timing and control of the task rather than polling. Interrupt handler disables interrupts and wakes the thread up. For flow control, transfer register empty (TXE) interrupt is used, which is built into USART [14]. After sending a byte, next byte is checked if it’s equal to zero (end of string). If it isn’t, TXE interrupt is enabled to wake the thread up once current byte has been sent. If next byte is zero, transmitting buffer index is reset and TXE interrupt is not enabled, thread will be woken up only by a receive register not empty (RXNE) interrupt or call from another task. If the thread is woken up by RXNE interrupt, byte is read from receive register and analyzed. If it’s beginning of message (BOM) byte, read register index is reset, if its end of message (EOM) byte, a string receive routine is called and string is passed to the routine. Code for the routine should be written by user of this API. Activity diagram about sending/receiving data through USART is shown in Figure 9.

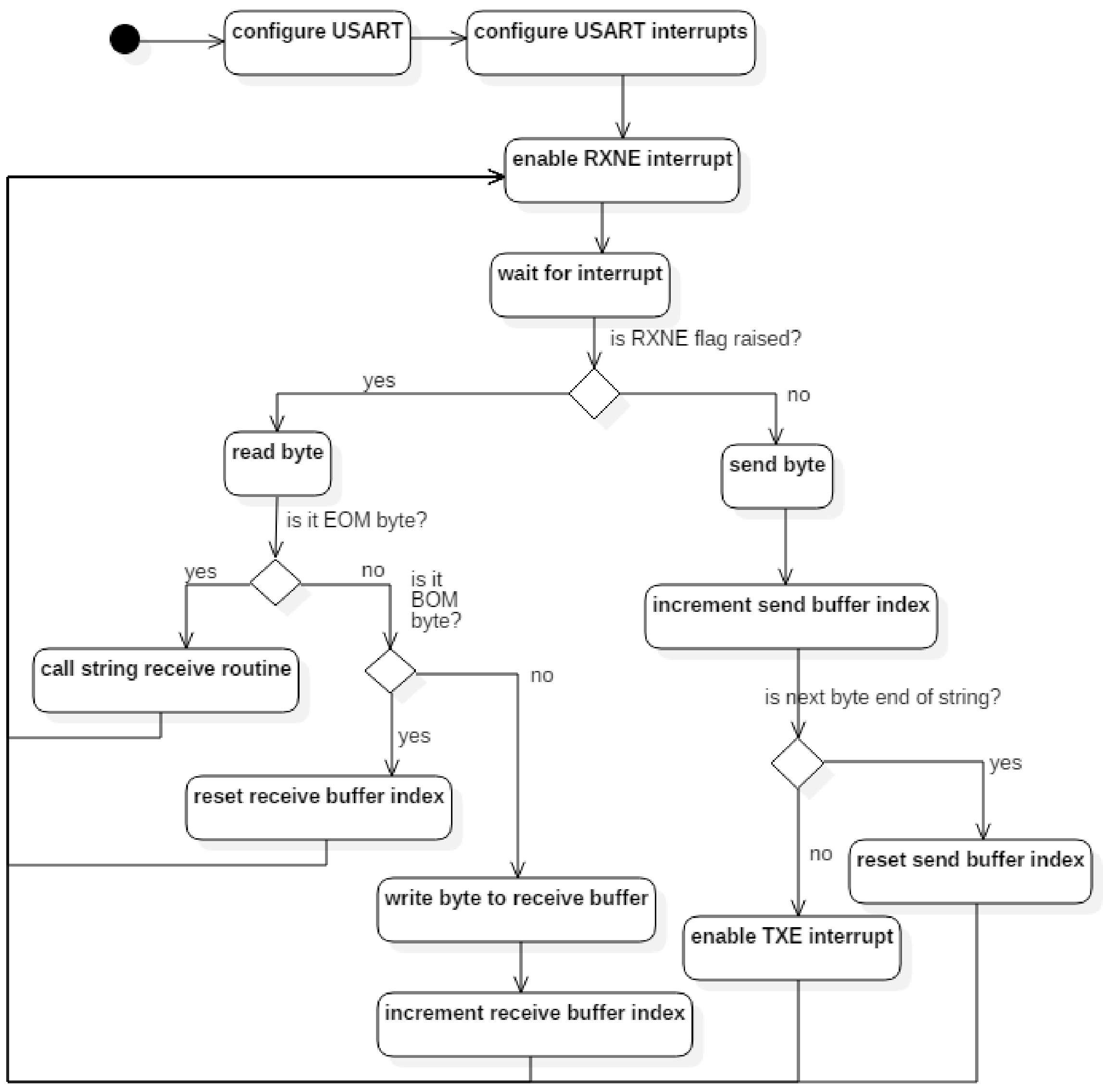
configure USART
configure USART interrupts
enable RXNE interrupt
wait for interrupt
is RXNE flag raised?
yes
no
read byte
send byte
is it EOM byte?
yes
no
is it BOM byte?
call string receive routine
no
yes
reset receive buffer index
write byte to receive buffer
increment receive buffer index
increment send buffer index
is next byte end of string?
yes
no
reset send buffer index
enable TXE interrupt

Figure 9. USART send/receive activity diagram. Author’s drawing

Because multiple different threads will be using the same thread for sending data, it has to be guaranteed that no data is lost and all gets transmitted in correct order. To do this, a large string is used a send buffer. When another thread wants to send a string of data while a string is already being sent, new string to be sent will be written to the end of current string that is being sent in the buffer, so that once send buffer index reaches end the of current string, new string will be sent right after.

4.2.2 API reference

Source code for this API is written directly for FreeRTOS, which means this will only work when this project is based on FreeRTOS. Because this code is based on interrupts, line where USART3's interrupt handler is needs to be replaced with **USART3_IRQHandler** in **startup.c** file in the project for this code to work. User of this API has following functions visible:

void vComPortTask(void *pvParameters) – this is function of the task that sends and receives data through USART3. The task needs to be created using **xTaskCreate()** provided by FreeRTOS's API before any other to be used.

extern void vStringReceiveRoutine(char []) – this function is called within **vComPortTask** when EOM byte has been received**.** Contents of this task have to be written user of this API, to parse received messages according to need. Parameter of this function is the string that has been received.

int comPrintf(const char* strMessage, ...) – this function passes a string to **vComPortTask** to be sent throuch USART3 to PC's com port. As mentioned earlier, this function is made thread-safe, which means this can be called by multiple different threads running in parallel without the risk of losing any data. The buffer where messages are queued, however, is not limitless which means sending a lot of data at once or spamming messages in a loop without delays is not advised. Returns 0 if writing the string to send buffer was successful, 1 if buffer was full. Parameters work the same way as for printf function contained within <stdio.h> library.

4.3 Gyroscope API

Gyroscope data can be used for testing and simulation of B-dot and attitude control algorithm on Earth, which is why gyroscope reading should be one of the first things to be done once communication with test board has been established. On STM32F3 test board, gyroscope is connected to the MCU through SPI interface [14]. On SPI interface, the master first sends out a byte which contains address and a bit which indicates whether read or write action is going to be performed [15].

4.3.1 API reference

This API is developed to work with L3GD20 gyroscope on board STM32F3 development board, but can be easily reconfigured to be used for a different device and/or SPI interface. Code for this API is written directly for FreeRTOS, which means this will work only with this OS.

Because the code works with interrupts, line where SPI1's interrupt handler is needs to be replaced with **SPI1_IRQHandler** within **startup.c** file of the project before using this API. User of gyroscope's API has following functions available for use:

uint8_t gyroReadFromRegister8(uint8_t address) – returns the 8 bit value from a register of the device connected to SPI. Parameter **address** is the 7-bit address of the register to be read. This function should not be called from two different threads running in parallel to avoid data corruption.

void gyro_WriteData(uint8_t address, uint8_t value) – writes an 8-bit value to addressed register. Parameter **address** is a 7-bit address of targeted register, parameter **value** is 8-bit value to be written to targeted register of the device. No return value.

void gyro_setup(void) – this function configures and initiates SPI1 interface and GPIO pins to which the gyroscope is connected to, configures interrupt vectors and finally powers up the gyroscope so that it's registers can be read. This function must be called before any of the above two functions are called or else the thread on which they are called will suspend and never wake up.

void gyro_SetDPS(char DPS) – sets gyroscope's degrees per second (DPS) setting. L3GD20 has three possible DPS settings: 250, 500 and 2000. Parameter **DPS** can be one of either three values contained within header file: L3GD20_DPS_200, L3GD20_DPS_500 or L3GD20_DPS_2000.

void gyro_powerup(void) – wakes gyroscope up from either sleep or power-down mode.

void gyro_powerdown(void) – puts gyroscope in power-down mode.

void gyro_sleep(void) – puts gyroscope in sleep mode.

5 Summary

Goal of this thesis was to develop software architecture of TUT nanosatellite’s ADCS subsystem. To accomplish this goal, based on TUT nanosatellite’s ADCS’s hardware design, requirements for software design were analyzed by examining previous student nanosatellite missions. In addition, datasheets of various hardware components were examined.

Technically most challenging parts were analysis and elaboration of the details of ADCS system and to understand and develop general software architecture where all smaller components are working together.

As a result of this work, requirements for software were gathered, main software components were specified, component interactions were defined, tasks within operating system were allocated, working cycle was specified and overall software architecture was developed. In addition to this, an application programming interface was developed for simulation purposes.

Practical value of this work is a comprehensive system analysis that makes simpler future organization and allocation of tasks inside the ADCS development team. A simple and universally applicable software API has been developed.

Bibliography

[1] TUT Mektory, “TUT - Mektory Nanosatellite programme,” TUT Mektory, [Online]. Available: http://www.ttu.ee/?id=115635. [Accessed 11 March 2016].

[2] R. Munakata, “CubeSat Design Specification,” California Polytechnic State University, San Luis Obispo, 2009.

[3] “SwissCube Phase A ADCS report,” Swiss Federal Institute of Technology in Lausanne, Lausanne, 2006.

[4] A. Slavinskis, U. Kvell, I. Sünter, H. Kuuste, S. Lätt, K. Voormansik and M. Noorma, “High spin rate magnetic controller for nanosatellites,” *ActaAstronautica,* vol. 218, no. 226, pp. 1-9, 2013.

[5] D. Bhanderi, B. Ø. Andresen, C. Grøn, R. H. Knudsen, C. Nielsen, K. K. Sørensen and D. Taagaard, “Attitude Control system for AAUSAT-II,” Aalborg University, Aalborg, 2005.

[6] ST Electronics, “STM32F3DISCOVERY - STMicroelectronics,” 21 04 2016. [Online]. Available: http://www2.st.com/content/st_com/en/products/evaluation-tools/product-evaluation-tools/mcu-eval-tools/stm32-mcu-eval-tools/stm32-mcu-discovery-kits/stm32f3discovery.html.

[7] J. B.-Y. Tsui, “Fundamentals of Global Positioning System Receivers: A Software Approach,” John Wiley & Sons, Inc, 2000.

[8] F. Jordan, “SwissCube Phase A Electrical Power System Final Report,” Swiss Federal Institute of Technology in Lausanne, Lausanne, 2006.

[9] E. Priidel, *Project's work group minutes,* Tallinn, 2015/2016.

[10] Real Time Engineers Ltd., “FREERTOS,” Real Time Engineers Ltd., [Online]. Available: http://www.freertos.org/. [Accessed 14 05 2016].

[11] B. Despont, “System Engineering and development and test of the ADCS breadboard for SwissCube,” Swiss Federal Institute of Technology in Lausanne, Lausanne, 2007.

[12] C. C. Finlay, S. Maus, C. D. Beggan, T. N. Bondar, A. Chambadut, T. A. Chernova, A. Chulliat, V. P. Golovkov, B. Hamilton, M. Hamoudi, R. Holme, R. Hulot, W. Kusang, B. Langlais, V. Lesur, F. J. Lowes, H. Lühr, S. Macmillan, M. Mandea, S. McLean, C. Manoj, M. Menvielle, I. Michaelis, N. Olen, J. Rauberg, M. Rother, T. J. Sabaka, A.

Tangborn, L. Tøffner-Clausen, E. Thébault, A. W. P. Thomson, I. Wardinski, Z. Wei and T. I. Zvereva, "International Geomagnetic Reference Field: the eleventh generation," *Geophysical Journal,* vol. 183, no. 3, pp. 1216-1230, 2010.

[13] R. A. Serway, Physics for Scientists and Engineers, Saunders College Publishing, 1986.

[14] ST Electronics, "STM32DISCOVERY Discovery kit for STM32F303xx microcontrollers," 02 2013. [Online]. Available: http://www.st.com/st-web-ui/static/active/jp/resource/technical/document/user_manual/DM00063382.pdf.

[15] ST Electronics, "L3GD20 Datasheet - production data," ST Electronics, Singapore, 2013.

[16] M. Rebane, "Progress report- TUT MEKTORY ADCS system," TUT, Tallinn, 2015.