



**A BENCHMARK FOR ASSESSING
NANOSATELLITE ON-BOARD COMPUTER
POWER-EFFICIENCY AND PERFORMANCE**

ANA CAROLINA CABRAL PIMENTEL DE MELO

**MASTER'S DEGREE THESIS PRESENTED IN THE POSTGRADUATE
PROGRAM IN ELECTRONIC SYSTEMS AND AUTOMATION
DEPARTMENT OF ELECTRICAL ENGINEERING**

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

**UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA**

**A BENCHMARK FOR ASSESSING
NANOSATELLITE ON-BOARD COMPUTER
POWER-EFFICIENCY AND PERFORMANCE**

ANA CAROLINA CABRAL PIMENTEL DE MELO

Supervisor: PROF. DR. RENATO ALVES BORGES, ENE/UNB

DISSERTAÇÃO DE MESTRADO EM ENGENHARIA ELÉTRICA

BRASÍLIA-DF, 16 DE DEZEMBRO DE 2019.

**UNIVERSIDADE DE BRASÍLIA
FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA**

**A BENCHMARK FOR ASSESSING
NANOSATELLITE ON-BOARD COMPUTER
POWER-EFFICIENCY AND PERFORMANCE**

ANA CAROLINA CABRAL PIMENTEL DE MELO

DISSERTAÇÃO DE MESTRADO ACADÊMICO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA, COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA.

APROVADA POR:

Prof. Dr. Renato Alves Borges, ENE/UnB
Orientador

Dr. Thyrso Villela Neto, CGEE/INPE
Examinador externo

Prof. Dr. Marcelo Grandi Mandelli, CIC/UnB
Examinador interno

BRASÍLIA, 16 DE DEZEMBRO DE 2019.

FICHA CATALOGRÁFICA

ANA CAROLINA CABRAL PIMENTEL DE MELO

A BENCHMARK FOR ASSESSING NANOSATELLITE ON-BOARD COMPUTER POWER-EFFICIENCY AND PERFORMANCE

2019xv, 147p., 201x297 mm

(ENE/FT/UnB, Mestre, Engenharia Elétrica, 2019)

Dissertação de Mestrado - Universidade de Brasília

Faculdade de Tecnologia - Departamento de Engenharia Elétrica

REFERÊNCIA BIBLIOGRÁFICA

ANA CAROLINA CABRAL PIMENTEL DE MELO (2019) A BENCHMARK FOR ASSESSING NANOSATELLITE ON-BOARD COMPUTER POWER-EFFICIENCY AND PERFORMANCE. Dissertação de Mestrado em Engenharia Elétrica, Publicação 736/19, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 147p.

CESSÃO DE DIREITOS

AUTOR: Ana Carolina Cabral Pimentel de Melo

TÍTULO: A BENCHMARK FOR ASSESSING NANOSATELLITE ON-BOARD COMPUTER POWER-EFFICIENCY AND PERFORMANCE.

GRAU: Mestre ANO: 2019

É concedida à Universidade de Brasília permissão para reproduzir cópias desta dissertação de Mestrado e para emprestar ou vender tais cópias somente para propósitos acadêmicos e científicos. O autor se reserva a outros direitos de publicação e nenhuma parte desta dissertação de Mestrado pode ser reproduzida sem a autorização por escrito do autor.

Ana Carolina Cabral Pimentel de Melo
ana.cpmelo95gmail.com

Agradecimentos

Primeiramente, gostaria de expressar minha eterna gratidão à minha família, que são o motivo da minha existência e que possibilitaram essa conquista, assim como todas as anteriores. Mãe, pai, vó, sem vocês eu nada seria. Obrigada pelo apoio, compreensão e amor que sempre me deram.

Gostaria de agradecer imensamente ao meu orientador, Prof. Renato, por essa oportunidade que mudou a minha vida e mais ainda, por ter me recebido de braços abertos, sempre muito prestativo, incentivando e inspirando o trabalho de todos dentro do laboratório. Sou extremamente grata aos professores Sandro Haddad e Daniel Café pela atenção, paciência e por compartilharem comigo um pouco do vasto conhecimento que possuem. Reconheço também a ajuda sempre presente dos professores parceiros do laboratório, Prof. Simone e em especial, Prof. Chantal, por ser uma inspiração como professora, mãe e engenheira em um domínio ainda predominantemente masculino.

Aos meus amigos da universidade, em especial Letícia, Luan, Lukas e Victor, que compartilharam o prazer e o estresse do dia-a-dia, meu muito obrigada. Aprendi muito com cada um de vocês e os recordarei para sempre com muito carinho e gratidão. Obrigada também às minhas amigas de infância, Ana Beatriz e Luisa, e aos parceiros da graduação, Lorena e Samael, por todo suporte emocional para concluir essa etapa da minha vida.

Aproveito para expressar ao meu orientador da graduação, Prof. Vincent Bourguet, que suas palavras de incentivo me acompanham até hoje. Por fim, gostaria de agradecer o apoio das entidades que possibilitaram a realização do presente trabalho: Universidade de Brasília, Fundação de Apoio a Pesquisa do Distrito Federal (FAPDF), Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) e Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq).

Abstract

This manuscript is aimed at specifying, implementing and executing a benchmark to evaluate the performance of different microcontrollers of nanosatellite onboard computers. The focus of this work is on Attitude Determination and Control System (ADCS) applications and therefore, a review of common attitude determination and control algorithms is presented in order to identify the main features to be explored. The benchmark proposal specifies a workload, which corresponds to a set of instructions that represent the application in question, a metric for evaluating the performance of different architectures and operational rules to ensure fair and reliable results. The implementation was carried out in a higher-level language, C, and it was validated by running it on four different architectures, where execution time and power measurements were used to evaluate energy consumption. The development platforms evaluated were the Texas Instruments MSP430FR5994 Launchpad Kit, STMicroelectronics Nucleo-L432KC board, Arduino Uno and Raspberry Pi 3B. The results obtained show that Arduino Uno has the highest energy consumption (1.41mJ) while Nucleo L432KC has the lowest (0.05mJ). These results were also analyzed with the purpose of using these platforms in the projects currently developed at the Laboratory of Simulation and Control of Aerospace Systems at the University of Brasilia where the Raspberry Pi was chosen for the applications of the nanosatellite simulator facility and the Nucleo L432KC for LAICAnSat activities.

Index terms: Onboard computer, power efficiency, nanosatellite, benchmark.

Resumo

O presente manuscrito tem como escopo a especificação, implementação e execução de um *benchmark* para avaliar o desempenho de diferentes microcontroladores de computadores de bordo de nanossatélites. O foco deste trabalho está nas aplicações dos sistemas de determinação e controle de atitude (ADCS, do inglês *Attitude Determination and Control System*), e portanto, uma revisão de algoritmos de determinação e controle de atitude utilizados em missões espaciais é apresentada, visando a identificação de recursos a serem explorados. A proposta do *benchmark* especifica uma carga de trabalho, que corresponde a um conjunto de instruções representativas da aplicação em questão, uma métrica para avaliar o desempenho de diferentes arquiteturas e regras operacionais para garantir resultados justos e confiáveis. A implementação foi realizada em linguagem de alto nível, C, e sua validação foi realizada através de sua execução em quatro diferentes arquiteturas, onde medidas de tempo e consumo de potência são utilizados para avaliar o consumo de energia. As plataformas de desenvolvimento avaliadas foram o MSP430FR5994 Launchpad Kit da Texas Instruments, a placa Nucleo-L432KC da STMicroelectronics, o Arduino Uno e o Raspberry Pi 3B. Os resultados obtidos mostram que o Arduino Uno apresenta o maior consumo de energia (1.41mJ) enquanto o Nucleo L432KC possui o menor (0.05mJ). Esses resultados também foram analisados visando a utilização dessas plataformas nos projetos desenvolvidos atualmente no Laboratório de Simulação e Controle de Sistemas Aeroespaciais da Universidade de Brasília, onde o Raspberry Pi foi escolhido para as aplicações do simulador de atitude de nanossatélites e o Nucleo L432KC para as atividades do LAICAnSat.

Palavras-chave: Computador de bordo, eficiência energética, nanossatélite, benchmark.

Contents

AGRADECIMENTOS	I
ABSTRACT	II
RESUMO	III
ACRONYMS	VIII
1 INTRODUCTION	1
1.1 SMALL SATELLITES.....	1
1.1.1 CUBESAT SUBSYSTEMS	3
1.2 WORKPLACE FRAMEWORK.....	5
1.2.1 LAICANSAT	6
1.2.2 THREE-AXIS NANOSATELLITE SIMULATOR FACILITY	8
1.3 OBJECTIVES	9
1.4 THESIS OUTLINE	9
2 THE NANOSATELLITE OBC BENCHMARK	11
2.1 WORKLOAD	11
2.1.1 OVERVIEW OF COMMON BENCHMARKS	14
2.2 THE BENCHMARK PROPOSITION	18
2.2.1 IMPLEMENTATION	18
2.2.2 METRICS AND RULES.....	19
2.3 TARGET DEVICES	20
2.3.1 MEASUREMENT PROCEDURES	22
2.4 SUMMARY	23
3 EXPERIMENTAL RESULTS	25
3.1 EXECUTION TIME	26
3.2 ENERGY CONSUMPTION	27
3.3 REFERENCE PLATFORM	30
3.4 LODESTAR APPLICATIONS	31
4 CONCLUSION	33

4.1	FUTURE WORKS	34
	REFERENCES	35
	REFERENCES	35
A	REFERENCE FRAMES.....	40
A.1	BODY-FIXED FRAME	40
A.2	LOCAL-VERTICAL/LOCAL-HORIZONTAL FRAME	41
A.3	EARTH-CENTERED-EARTH-FIXED	41
A.4	EARTH-CENTERED INERTIAL	42
B	ATTITUDE REPRESENTATIONS	43
B.1	DEFINITIONS	43

List of Figures

1.1	Common CubeSat Configurations [3].	2
1.2	Poly-Picosatellite Orbital Deployer [5].	2
1.3	Usual CubeSat architecture.	4
1.4	LAICAnSat mission lifecycle.	6
1.5	LAICAnSat developments.	7
1.6	Pressure valve prototype [19].	8
1.7	Helmholtz cage.	9
2.1	Generic setup of measurement.	20
2.2	Setup of measurement.	22
3.1	Power consumption per MHz (mW/MHz).	27
3.2	Energy consumed (mJ).	28
3.3	Power consumption (mW/MHz) per function.	29
3.4	EnergyTrace++ comparison.	30
A.1	3U CubeSat body-fixed frame.	40
A.2	Local-Vertical/Local-Horizontal Frame.	41
A.3	ECEF reference frame.	41
A.4	ECI reference frame.	42
B.1	Roll (ϕ), pitch (θ) and yaw (ψ) angles.	44

List of Tables

1.1	Small satellite classification by mass.	1
1.2	Common benchmarks.	5
1.3	Embedded benchmarks.....	5
2.1	Attitude representations summary.	12
2.2	Benchmark descriptions.....	15
2.3	BEEBS benchmarks.	17
2.4	Embedded benchmarks description.	17
2.5	Commercial onboard computers microprocessors (adapted from [?, ?]).....	21
2.6	Hardware specifications.....	21
3.1	Code size.	26
3.2	Elapsed time.	26
3.3	Power consumption.	27
3.4	Power and energy consumption.	28
3.5	Results related to a reference platform	31
3.6	Hardware peripherals.	32

Acronyms

ADA	Analysis, Design and Algorithm
ADC	Analog to Digital Converter
ADCS	Attitude Determination and Control System
ALGOL60	Algorithm Language 1960
AO	Atomic Oxygen
BEEBS	Bristol Energy Efficiency Benchmark Suite
BLAS	Basic Linear Algebra Subprograms
C&DH	Command and Data Handling
CDS	CubeSat Design Specifications
COTS	Commercial-Off-the-Shelf
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
DEC	Digital Equipment Corporation
DMA	Direct Memory Access
DMIPS	Dhrystone Million Instructions per Second
EEMBC	Embedded Microprocessor Benchmark Consortium
EPS	Electrical Power System
ESQ	Estimators of the Optimal Quaternion
FLOPS	Floating-point Operations Per Second
FOAM	Fast Optimal Attitude Matrix
FPU	Floating-point Unit
GCC	GNU Compiler Collection
GPIO	General Purpose Input/Output
GPU	Graph Processing Unit
HDMI	High-Definition Multimedia Interface
IDE	Integrated Development Environment
LEO	Low Earth orbit
LODESTAR	Laboratory of Simulation and Control of Aerospace Systems
LPM	Low Power Mode
MCU	Microcontroller Unit
OBC	Onboard Computer
P-POD	Poly-Picosatellite Orbital Deployer
PID	Proportional-Integral-Derivative
PWM	Pulse Width Modulation

QUEST	Quaternion Estimator
RAM	Random Access Memory
ROM	Read Only Memory
SO(n)	Special Orthogonal group
SPEC	Standard Performance Evaluation Corporation
SPI	Serial Peripheral Interface
TRIAD	Triaxial Attitude Determination
TT&C	Telemetry, Tracking and Command
UHV	Ultra-high Vacuum
ULP	Ultra-low Power
UnB	University of Brasília
USB	Universal Serial Bus
UV	Ultraviolet
WIPS	Whetstone Instructions Per Second

Chapter 1

Introduction

1.1 Small satellites

The interest in space exploration dates back to the dispute between the United States and the Soviet Union in search of pioneering access to space. This dispute, known as the Space Race, was marked by the launch of the first artificial satellite, Sputnik I, in 1957. Since then, the magnitude of space missions has increased, thereby increasing substantially the complexity and size of spacecrafts, as well as development time. All of these factors have limited access to space to only a few wealthy companies and government agencies with large financial budgets and technically specialized teams.

In this context, the emergence of small satellite technologies has become an interesting approach for rapid and democratic access to space. The small satellite category encompasses the spacecrafts with total mass below 180kg, classified according to Table 1.1 [1].

Classification	Mass (kg)
Mnisatellite	100-180
Microsatellite	10-100
Nanosatellite	1-10
Picosatellite	0.01-1
Femtosatellite	0.001-0.01

Table 1.1: Small satellite classification by mass.

In particular, a specific class of nanosatellites, the *CubeSats*, leveraged this new mindset of developing space technology cheaper, faster and more independently. The concept was created by Prof. Robert Twiggs and Prof. Jordi Puig-Suari at Stanford University and California Polytechnic State University [2], initially thought as an educational tool for university students to learn about the process of designing a space mission through hands-on experience. The development cycle should be short enough to be completed during an undergraduate course, and this has been achieved through miniaturization of real aerospace systems

with simplified functions and the use of commercially available technology. A CubeSat, as the name implies, is a cubic-shaped miniaturized satellite with edges of 10cm. Each cube represents a unit (1U) that can be stacked in two or more to form CubeSats of up to 24U, with a limited weight of 1.33kg per unit. Figure 1.1 shows common CubeSat configurations.

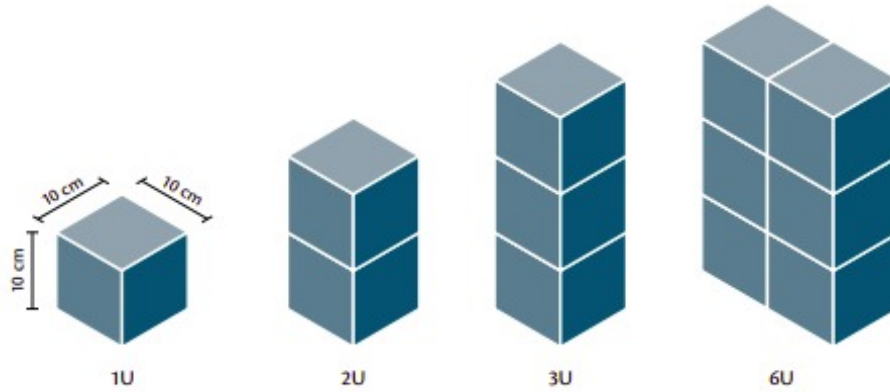


Figure 1.1: Common CubeSat Configurations [3].

Nowadays, this new concept of designing a space mission is widely spread, even beyond universities, being adopted by private companies and government agencies in a variety of applications, such as remote sensing, technology demonstration, communications and others [3, 4]. It complies with an open-source specification of basic physical features and safety requirements, the CubeSat Design Specification (CDS) [5].

In general, this satellite class is known for being released as secondary payloads aboard larger missions, and the CDS benefited both developers, by optimizing launch costs even further, and launchers, by ensuring the integration of the modules and safety of the launch vehicle. Shown in Figure 1.2 is the Poly-Picosatellite Orbital Deployer (P-POD), the first design meant to release on orbit three CubeSats as secondary payloads of larger spacecrafts.

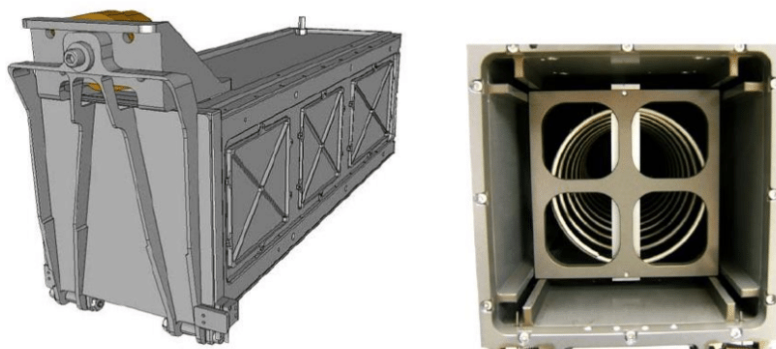


Figure 1.2: Poly-Picosatellite Orbital Deployer [5].

1.1.1 CubeSat subsystems

Although it is a miniaturized version, a CubeSat follows the architecture of large space systems, which comprises the following subsystems, each one developing its specific functionality. Figure 1.3 shows the generic architecture, adapted from [6, 7]. A summary of the main subsystems is presented below in accordance with [8].

- *Command and Data Handling (C&DH) system*: it gathers, processes and formats housekeeping and mission data and manages communications between other systems. This functionality is usually performed by an onboard computer (OBC), that also comprises the memories (boot, safeguard and work memories), data buses and bus controllers, debug interface and real-time clock.
- *Attitude Determination and Control System (ADCS)*: as the name implies, it is responsible to determine the attitude of the spacecraft and control its orientation. It uses actuators such as reaction wheels or magnetic torque rods to stabilize the vehicle or to achieve a required maneuver, e.g. point solar panels towards the sun or antennas towards the Earth. In order to do so, it collects data from sensors, such as star trackers, sun sensors, magnetometers, gyroscopes, and others to determine its orientation relative to some reference frame.
- *Telemetry, Tracking and Command (TT&C) system*: it is the interface between the spacecraft and the ground station. It allows sending and receiving information both ways, as well as tracking the location of the satellite. This information can be telemetry data to inform about the health of the satellite and/or commands from the ground station to execute a specific function.
- *Electrical Power System (EPS)*: it is responsible for providing a stable and continuous power source. That means it conditions, stores and distributes the electrical power to all spacecraft systems, as well as protects the electronics from non-nominal voltages and currents. It consists mainly of a power source, energy storage, regulation and control units, and power distribution.
- *Payload*: it consists of the necessary hardware to perform the intended satellite mission. It may be remote sensing, imaging, technology demonstration, science or others.

Regarding the computer system architecture, it can be centralized, distributed or a combination of both (hybrid) [7, 9, 10]. In the first, there is one main processing unit with point-to-point interfaces between the other ones, achieving a highly reliable topology if employed with few subsystems. In the second one, there is more than one processor sharing a common

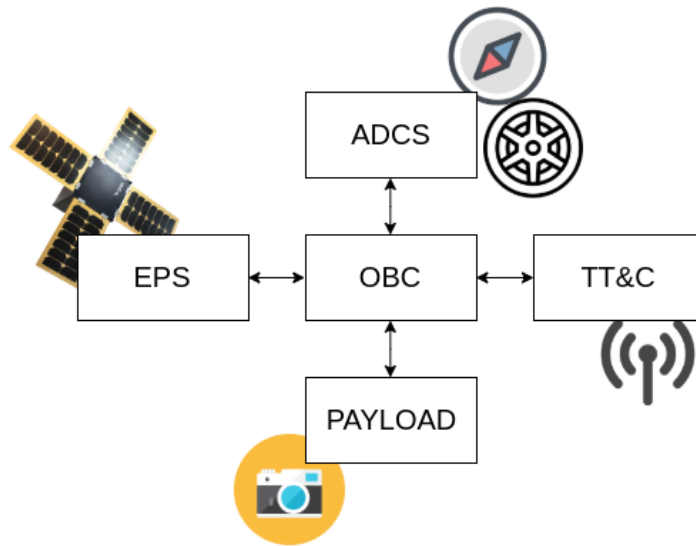


Figure 1.3: Usual CubeSat architecture.

data bus, which is very useful for implementing fault tolerance and redundancy with limited impact on size, weight and power. In general, a hybrid approach is used when there are systems that require critical processing and others that can rely on one single processor unit to perform their tasks.

Moreover, in the case of nanosatellites, due to their limited size, weight and power constraints, multiple processing tasks are often incorporated into a single onboard computer, such as ADCS and C&DH functionalities. This type of space mission is characterized by a reduced-lifetime (less than five years) at a Low Earth Orbit (LEO), making the use of commercial-off-the-shelf (COTS) processors more attractive than ad-hoc¹ development.

Throughout the computer system design, it is important to have a quantitative mechanism for comparing computer resources. A common approach to optimize a design process is to use a benchmark since it provides a common method of comparing different systems across different architectures by simulating a particular type of workload on a component or system. It prevents the designer from making oversized or undersized choices, saving customers money and development time.

A benchmark program should contain a set of relevant instructions that reflects the final operation. According to [10], the software for onboard processing can be classified as control system software, system management software, command and data handling software, payload management and operating system software. The focus of this work relies on the control system software, which is mathematically intensive and performs many floating-point computations. Also, a metric is used to evaluate different devices. Due to limited power constraints, in the case of nanosatellites it is interesting to have a metric to evaluate energy consumption as well. In Table 1.2 the workloads of some common benchmarks are described, however these are not designed for embedded applications.

¹For a particular purpose or need.

Name	Description
Whetstone	It evaluates performance on floating-point operations.
Linpack	It evaluates basic vector and matrix operations with floating-point calculations.
Dhrystone	It addresses string handling and integer operations.
SPEC	Comprises a set of programs to evaluate performance of computing systems.
JouleSort	It measures the amount of energy required to sort 10^{10} records.

Table 1.2: Common benchmarks.

In fact, the embedded domain is led by a consortium that comprises all the main suppliers and provides suites for different applications, the Embedded Microprocessor Benchmark Consortium (EEMBC), which concentrates most of the work developed in this area. Table 1.3 presents benchmarks focused on providing small programs. Only EEMBC and BEEBS provide energy measurements, but the EEMBC benchmarks are not readily accessible due to the high cost to adhere to the consortium and the workloads from BEEBS do not exploit the elementary operations of a control system software.

Name	Description
EEMBC	Benchmark suites for various applications, such as mobile devices and many others.
MiBench	Benchmark suites for various applications, following EEMBC model.
BEEBS	Set of small programs chosen from available suites.
MLPerf	Machine learning (training and inference) benchmarks.
EmBench	Set of small programs chosen from available suites.

Table 1.3: Embedded benchmarks.

A detailed overview of these benchmarks is presented in Chapter 2. The proposal of a new benchmark is justifiable as no set of programs was found to evaluate the desired functions.

1.2 Workplace Framework

Considering the capability to reduce costs of space mission by using the small and standardized CubeSat platforms, the Laboratory of Simulation and Control of Aerospace Systems (LODESTAR) at the University of Brasilia (UnB) has embraced this opportunity and has been working towards the establishment of a nanosatellite program at UnB. Although the

first CubeSat mission, the Alfa Crux I, has just started at LODESTAR, preliminaries activities and projects have paved the way to get a simple concept of a nanosatellite to a complete operational spacecraft in orbit. The two main projects that provide the first experience with the process of a CubeSat mission and the development of specific subsystems such as the ADCS, are the LAICAnSat project, a high altitude platform for stratospheric experiments and applications, and a nanosatellite three-axis simulator facility. A brief overview of both projects is provided in the sequel in order to assess the goals of both initiatives. The idea is to apply the proposed benchmark to help decide whether their objectives are achievable with regard to OBC performance and power consumption.

1.2.1 LAICAnSat

The LAICAnSat project started in 2013. It aims to develop a vehicle capable of conducting scientific studies at high and low altitudes. Meteorology and remote sensing applications are among some of the applications intended, as it is the qualification of commercial-off-the-shelf technology and validation of control algorithms for landing systems [11, 12].

Figure 1.4 shows a mission lifecycle, where a platform is sent at an altitude of up to 30km through an atmospheric balloon. Traditionally, latex balloon bursts at a certain altitude and releases the payload, which falls in a free flight with a parachute to cushion the impact and it is recovered upon reaching the ground.

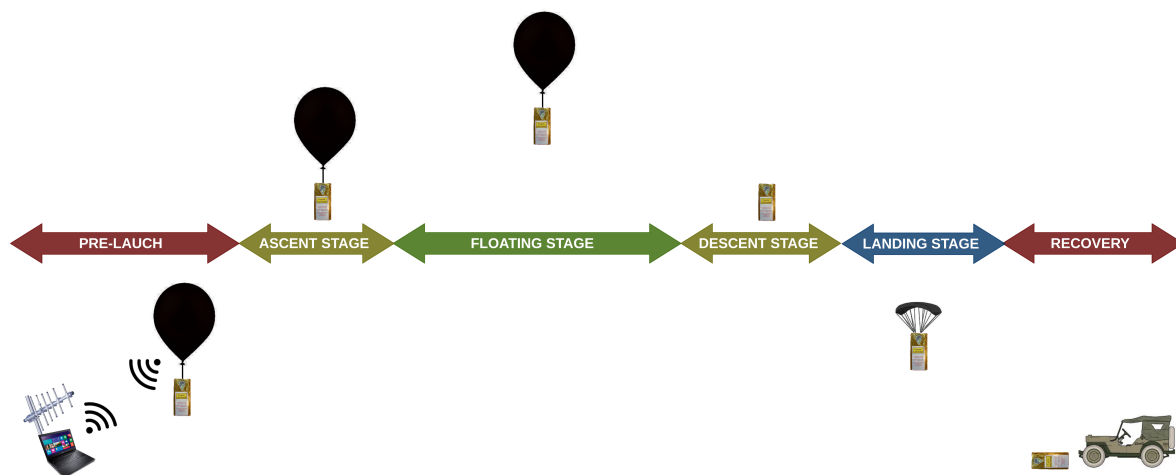
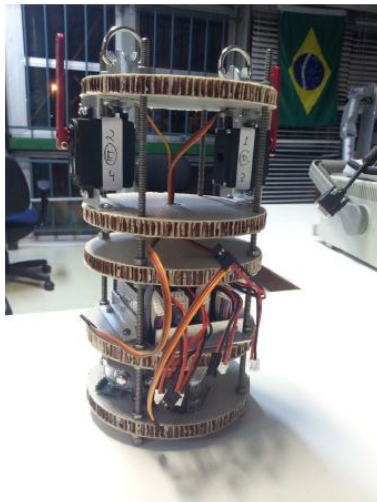


Figure 1.4: LAICAnSat mission lifecycle.

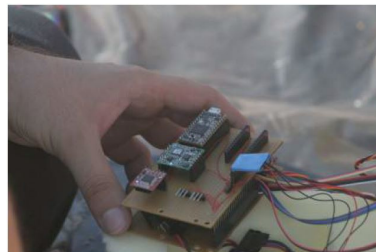
The first prototype was a TubeSat shape platform [13] equipped with a high-resolution camera and a sensor suite that provided temperature, humidity, pressure, UV light level and power informations. The first electronic system consisted of two modules: one for data acquisition responsible for processing and storing data from the sensors and another for attitude determination and communication. The components were connected using a breadboard and were arranged inside a styrofoam box that provided thermic isolation. The payload was a camera that worked independently by storing time-lapse images on an internal

memory card.

After two successful missions (LAICAnSat-1 and 2) have validated the first assumptions adopted in the project, further enhancements included a structural standardization, manufacturing method and researches on trajectory control [14, 15, 16]. The structure adopted the CubeSat standard fabricated using rapid prototype technologies and the PC/104² standard was chosen as a guideline for embedded systems [17]. Figure 1.5 shows both structures and their respective electronic systems.



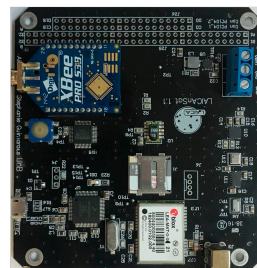
(a) TubeSat structure.



(b) First electronics.



(c) CubeSat structure.



(d) Standardized OBC.

Figure 1.5: LAICAnSat developments.

With this standardized system, the team succeed in its fifth launch, which was concerned with the recording of the total solar eclipse from the stratosphere in 2017 within the NASA Space Grant Eclipse Ballooning Project [18], and marked the beginning of new steps for the project. Based on the analysis of previous missions, several aspects related to the system development and its modules integration were identified in order to be improved. Also,

²PC/104 defines both form factors and computer buses to make the printed-circuit board modular and stackable.

following the evolutionary scale of this project, enhancements are needed in order to provide a better service as means of conducting these experiments. Therefore, the entire system is being rethought to increase its capacity and reliability.

An example of an additional demand currently pursued is a floatation stage that is being planned to ensure scientific operations at a desired altitude. A control system equipped with a pressure valve is being developed in-house using rapid prototype technologies to be able to control the helium flow rate of the atmospheric balloon. Figure 1.6 shows the latest prototype, a spherical valve commonly used in many commercial applications and a servo motor to control its opening [19].

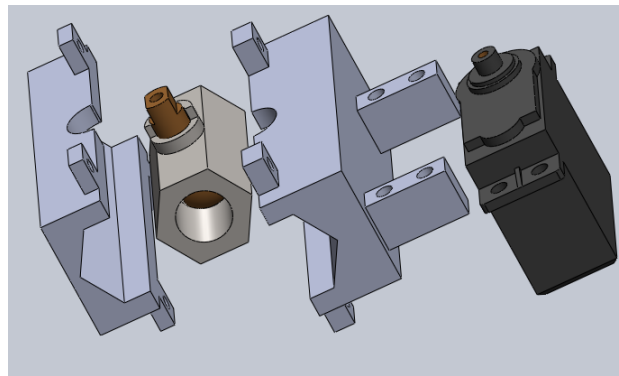


Figure 1.6: Pressure valve prototype [19].

Another required improvement concerns the payload recovery. As the balloon rises freely, there is a possibility that it will drift far away from the starting point. Without a landing control, the platform can land in places hard to reach, as it has already happened in previous missions. This led to another project front in charge of studying and developing parachute control methods [11, 12, 20, 21].

1.2.2 Three-axis nanosatellite simulator facility

Simultaneously to LAICAnSat activities, another project being developed at LODESTAR is a nanosatellite three-axis simulator facility [22, 23, 24, 25]. The purpose is to allow the simulation, testing and validation of attitude determination and control algorithms [26, 27, 28], in addition to other spacecraft technologies, such as magnetic torque rods [29, 30], reaction wheels and communication modules.

The simulator main components are an air-bearing table and a Helmholtz cage. A pneumatic system is used to replicate low gravitational torque and the frictionless conditions encountered by satellites in space [31]. The Helmholtz cage, shown in Figure 1.7, is a structure composed of three orthogonally disposed pairs of square coils used to replicate magnetic fields perceived by satellites in orbit. It is able to simulate variations in magnitude and direction from the generation of three-axis magnetic fields of intensities $\pm 1.5G$ with closed loop control. The nanosatellite is mounted on the air-bearing table that moves freely on top of the

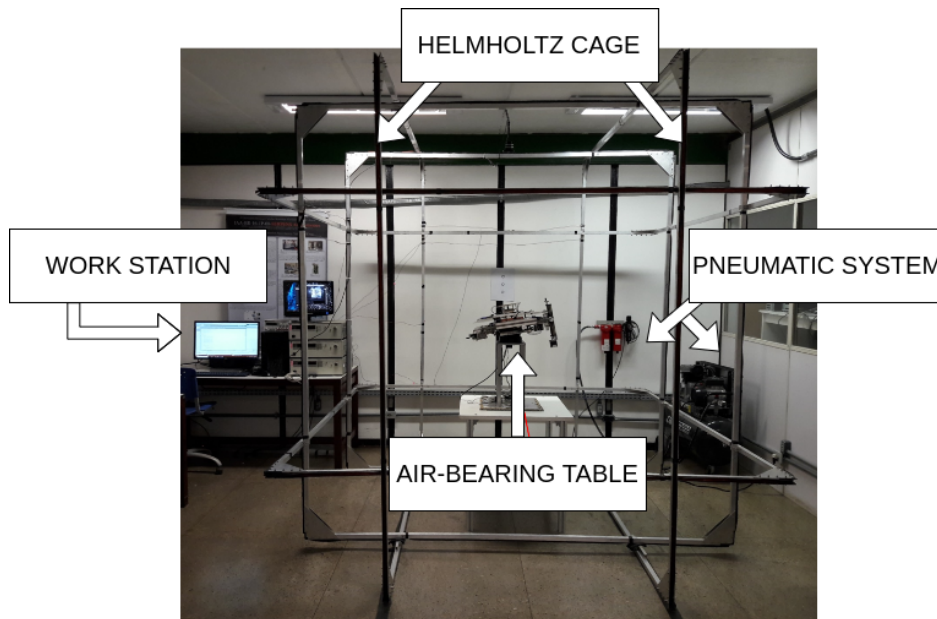


Figure 1.7: Helmholtz cage.

sphere around the yaw axis, but with roll and pitch angles limited to $\pm 45^\circ$.

1.3 Objectives

The primary objective of this work is to develop a benchmark based on applications of ADCS running on COTS OBC for nanosatellites. From that, specific objectives are listed as follows:

1. To propose a workload based on nanosatellites control system software. For that, the main ADCS algorithms currently used in space missions are analyzed to specify a relevant set of functions;
2. To select a metric to evaluate different microcontrollers for nanosatellites onboard computers;
3. To define an appropriate set of rules for the benchmark;
4. To validate the benchmark on different COTS microprocessors;
5. To illustrate its functionality and applicability in the context of the current projects being developed at LODESTAR.

1.4 Thesis outline

This manuscript is structured according to the following chapter distribution, including this introductory chapter concerned with introducing the objectives and motivation.

Chapter 2 presents a literature review on benchmarking and its applications in the embedded systems domain. In addition, the chapter also specifies the workload, metrics and rules of the benchmark in question. Furthermore, the hardware platforms chosen for validating the implementation are exposed.

Chapter 3 shows the results obtained and finally, the concluding remarks and future works perspectives are presented in Chapter 4.

Chapter 2

The nanosatellite OBC benchmark

A benchmark can be synthetic or application-based [32]. Synthetic benchmarks are artificially created to evaluate a particular component capability, like a hard disk or networking device, but do not reflect an user actual experience with the machine. Application-based benchmarks mimic a real-world workload by performing common functions from within a particular industry segment and measure the overall performance of a system. There is also a combination of both, called hybrid or derived benchmarks.

The conception of a benchmark must specify a workload that reflects a real application, a metric for comparison and rules to ensure that the benchmark operates in fair conditions [33]. To assign a proper workload one should keep in mind the scenario in which this benchmark will be evaluated, i.e on nanosatellite onboard computers.

Moreover, space missions require certain requirements to be met to ensure mission success, such as power consumption. This further restricts the specification of the desired benchmark in terms of metrics, as the most common is to evaluate only performance. One review of available benchmarks was conducted in order to identify one applicable to the context of the control system software and this discussion is presented throughout this chapter.

Then, the methodology adopted to implement the workload and a relevant metric to compare different machines are presented. Latter, the real hardware platforms used for validating the benchmark are specified, as well as the procedures for running and measuring it on each device.

2.1 Workload

As mentioned before, the proposed benchmark is developed based on applications of ADCS running on COTS OBC for nanosatellites. In this sense, the main attitude estimation and control methods used nowadays are listed and analyzed as the basis for the workload choice. The goal is to define a representative set of instructions for numerical programming of such algorithms. Specifically, the main set of mathematical operations should be defined

Attitude Representations	
Euler axis (\mathbf{e}) and angle (ϑ)	$A(\mathbf{e}, \vartheta) = I_3 - \sin(\vartheta)[\mathbf{e}\times] + (1 - \cos(\vartheta))[\mathbf{e}\times]^2$
Rotation vector	$A(\mathbf{e}, \vartheta) = \exp([\vartheta\mathbf{e}\times])$
Quaternion ($\mathbf{q} = [\mathbf{q}_{1:3} \ q_4]^T$)	$A(\mathbf{q}) = (q_4^2 - \ \mathbf{q}_{1:3}\ ^2)I_3 - 2q_4[\mathbf{q}_{1:3}\times] + 2\mathbf{q}_{1:3}\mathbf{q}_{1:3}^T$
Rodrigues parameters ($\mathbf{g} = \frac{\mathbf{q}_{1:3}}{q_4}$)	$A(\mathbf{g}) = I_3 + 2\frac{[\mathbf{g}\times]^2 - [\mathbf{g}\times]}{1 + \ \mathbf{g}\ ^2}$
Modified Rodrigues parameters ($\mathbf{p} = \frac{\mathbf{q}_{1:3}}{1+q_4}$)	$A(\mathbf{p}) = I_3 + \frac{8[\mathbf{p}\times]^2 - 4(1 - \ \mathbf{p}\ ^2)[\mathbf{p}\times]}{(1 + \ \mathbf{p}\ ^2)^2}$
Euler angles (ϕ, θ, ψ)	$A_{ijk}(\phi, \theta, \psi) = A(\mathbf{e}_k, \psi)A(\mathbf{e}_j, \theta)A(\mathbf{e}_i, \phi)$

Table 2.1: Attitude representations summary.

in such a way that it fairly represents the computational complexity of the current ADCS algorithms. For that, one should keep in mind the different ways to represent the satellite attitude, the respective kinematics and dynamics equations, and the attitude determination and control methods themselves.

The attitude of a rigid body is estimated by comparing two reference frames, or two vectors [34, 35]. It is a linear transformation represented by a 3×3 matrix that rotates one reference frame into the other (alias sense) [36]. This matrix, called rotation matrix or attitude matrix, represents the physical attitude of the rigid body. It is well known that the set of all rotation matrices is the special orthogonal group of rigid rotations in \mathcal{R}^3 , denoted by $SO(3)$ (i.e. proper orthogonal matrix). Different representations may be used for the attitude matrix. Further informations about reference frames and attitude representations can be found on appendices A and B. Table 2.1 summarizes the main attitude representations.

The fundamental equation of attitude kinematics is obtained by evaluating the time derivative of the attitude matrix, and the parameterization follows from those appeared in Table 2.1. In other words, the time rate of change of a rigid body attitude is given by Equation 2.1:¹

$$\dot{A} = A[\omega\times]. \quad (2.1)$$

where $A \in SO(3)$ is the attitude matrix and $[\omega\times]$ is the cross product skew symmetric matrix obtained with the angular velocity vector ω . For the dynamics, first the moment of inertia tensor is defined, then the angular momentum and rotational kinetic energy are expressed in terms of the moment of inertia and angular velocity, given by Equation 2.2:¹

$$J\dot{\omega} = -[\omega\times]J\omega + T. \quad (2.2)$$

where J is the moment of inertia tensor and T the net external torque (control torque and disturbance torques due to, for instance, gravitational, aerodynamic and solar radiation). Once again, any representation may be used to parameterize the attitude motion [37, 36, 38].

Concerning the control laws and filter implementation for ADCS in modern-day, they are still very simple and well-known solutions, but extremely effective [39, 40, 36]. Attitude determination (static) is mainly based on the TRIAD algorithm that uses body and reference

¹The reference frames information are omitted for simplicity of notation.

frames observations to estimate the attitude matrix. The basic idea is as follows. If one has an orthonormal right-handed triad of vector $\{\mathbf{v}_1 \mathbf{v}_2 \mathbf{v}_3\}$ in the reference frame, and the corresponding orthonormal right-handed triad $\{\mathbf{w}_1 \mathbf{w}_2 \mathbf{w}_3\}$ in the body frame, then the attitude matrix is given by Equation 2.3 [36]:

$$A = [\mathbf{w}_1 \mathbf{w}_2 \mathbf{w}_3][\mathbf{v}_1 \mathbf{v}_2 \mathbf{v}_3]^T. \quad (2.3)$$

Improvements and extensions can be done in different ways leading to, for instance, the formulation of attitude estimation as a least square problem (Wahba's problem). The Wahba's problem provides the basis of most of the popular methods such as Quaternion Estimator (QUEST), Estimators of the Optimal Quaternion (ESOQ), and Fast Optimal Attitude Matrix (FOAM), and can be solved using quaternion parameterization through the Davenport's method [36, 38]. When it comes to attitude estimation, the Kalman filter and its extensions are widely used for onboard attitude estimators [41]. From the other side, most of the attitude controllers are simple proportional–integral–derivative (PID) solutions [42], although the use of Lyapunov-based methods has increased recently.

The interesting feature to be noticed in all of these methods, not only the solutions for attitude determination but also for control, is that all of them rely on fundamental numerical matrix operations, such as multiplicative, inverse, summation and difference. Not surprisingly, matrix product is among the most studied computational problems, belonging to the \mathcal{P} -class, that is, the algorithm computes the product in at most An^c steps for positive constants A and c with n being the matrix order [43, 44].

Nevertheless, it is well known that some problems in systems and control fall into the non-polynomial class (NP -class), as, for instance, the stability of interval matrix, very common in robust analysis and design [45]. Despite the nature of the problem, the set of instructions to be implemented (namely, elementary operations) will fall into the efficient computation category, that is, the ones whose runtime on any input of length n is bounded by a polynomial function in n (P -class). For instance, the inversion of integer matrices can be carried out using Gaussian elimination with $O(n^3)$ arithmetic operations.

In fact, many questions about linear systems can be decided efficiently, for example, controllability, stability, and observability of continuous or discrete time linear systems can be decided in polynomial time [45]. The theory of dynamical systems and the development of system analysis considering robustness and control led to classical, and in some cases new, matrix problems. Nowadays, a great number of different methods for stability analysis and controller and filter design are based on a matrix approach [46]. One of the fundamental ideas behind the Lyapunov theory, expected to be the future of spacecraft control designs, is that the stability of a dynamic system can be analyzed through the spectral properties of the system matrix [47]. Different concepts concerned with linear dynamical systems analysis, such as multiplicative and additive D-stability, diagonal stability, Schur D-stability, H-stability, depend on a stability region, a matrix class and an operation [46].

As a consequence, considering the application of the proposed benchmark and the discussion presented so far, the choice of the workload should definitely include a set of matrix operations instructions, specifically matrix multiplication and inversion. An overview of available benchmarks is presented next in order to identify one applicable to this context.

2.1.1 Overview of common benchmarks

The Whetstone benchmark was the first general purpose benchmark to set industry standards for computer system performance [48]. It is a synthetic benchmark written in algorithmic language 1960 (ALGOL 60) in 1972 by Dr. B. A. Wichman and Harold Curnow [49] to evaluate performance on floating-point operations of scientific computers. It also addressed the issue of the efficiency of the different programming languages, being latter translated to Fortran, C/C++, Basic, Visual Basic and Java. The results were given in thousands of Whetstone instructions per second (kWIPS), that eventually evolved to millions (MWIPS) with advances in technology. It was still run by Digital Equipment Corporation (DEC) and Intel until 1996 [50].

Dhrystone is a similar synthetic benchmark for integer and string operations. It was published in 1984 by R. P. Weicker [51], written in analysis, design and algorithm language (ADA). Its C version has become a key standard benchmark with the growth of Unix systems [52]. Nowadays, there are modified versions to run on Android tablets and phones and ARM CPUs, such as Raspberry Pi. Original versions of the benchmark provided Dhrystones per second performance ratings, which later was changed to VAX MIPS (DMIPS).

Similarly, around the same decade, the Linpack [53] was written by Jack Dongarra, Jim Bunch, Cleve Moler, and Gilbert Stewart. It was initially a package of Fortran subroutines for solving dense linear equations using Basic Linear Algebra Subprograms (BLAS) libraries. It evaluates the speed of supercomputer performing basic vector and matrix operations with 64-bit floating-point calculations, measured in floating-point operations per second (FLOPS).

The Standard Performance Evaluation Corporation (SPEC), founded in 1988, provides a standardized suite of benchmarks based on real applications divided into separate integer and floating-point categories. According to [54], SPEC is a "non-profit corporation formed to establish, maintain and endorse standardized benchmarks and tools to evaluate performance and energy efficiency for the newest generation of computing systems".

Among its benchmarks, SPEC released in 2007 the *SPECpower_ssj2008* [55, 56]. It aims to measure the power and performance characteristics of volume server-class computer equipment, exercising the CPUs, memories and other aspects of the operating system. Other benchmarks incorporate power measurements as well², but none is designed for embedded systems.

JouleSort [33] is an energy-oriented benchmark that provides the amount of energy re-

²Available at <https://www.spec.org/benchmarks.html#power>.

quired to sort 10^{10} records. It sorts a fixed number of randomly permuted 100-byte records with 10-byte keys from an input file. This workload is said to be simple and portable, but the way it was implemented requires access to a filesystem and manages a large amount of data (greater than 10GB), often incompatible with embedded applications.

Table 2.2 exhibits information regarding the previous benchmarks and their initial targets and developer entity, as well as the cost to obtain their source code (if freely available or required paid membership).

	Whetstone	Linpack	Dhrystone	SPEC	JouleSort
Year	1972	1977	1984	1989	2007
Initial target	Scientific computers	Supercomputers	System programming	Unix server	Data centers
Type	Synthetic	Library	Synthetic	Applications	Energy-oriented
Availability	Free	Free	Free	Paid membership	Free
Developer	Academia + Industry	Academia	Academia	Academia + Industry	Academia

Table 2.2: Benchmark descriptions.

All those benchmarks are not easily portable or even representative of small embedded systems, where there are particularities like small memory size and lack of operating systems. In 1997, the embedded community formed the Embedded Microprocessor Benchmark Consortium (EEMBC) to provide an industry standard for evaluating embedded system implementations. It offers a variety of suites that reflect real-world applications from five embedded markets: Automotive/Industrial, Consumer, Networking, Office Automation and Telecommunications [57, 58].

The benchmarks can be grouped into Ultra-Low-Power and Internet of Things, Heterogeneous Computer, Phone and Tablet or Single-core or Multi-core Performance categories. Its Ultra-Low Power (ULP) suite is designed to run on devices from 8- to 64-bit microprocessors and it has three profiles:

- *ULPMark-CoreProfile*: consists of commonly used functions, such as memory and math operations, sorting, GPIO interaction, besides others. It focuses on the MCU's core and the transition between low-power and active modes;
- *ULPMark-PeripheralProfile*: focuses on commonly used peripherals, such as Real Time Clock (RTC), Pulse-width Modulation (PWM), Analog-to-Digital Conversion (ADC) and Serial Peripheral Interface (SPI);
- *ULPMark-CoreMark*: is comprised of list processing (find and sort), common matrix manipulation and state machine processing based on switch and if statements. A Cyclic Redundancy Check (CRC) is also implemented to ensure the accuracy of the outputs, besides being a commonly used workload itself.

The EEMBC consortium leads the industry but it is only available upon paid membership.

Similar to EEMBC suites, there are the MiBench suites [59] that follow the same model. It is freely available as standard C source code and is divided into six categories:

1. Automotive and Industrial Control: it evaluates basic math abilities, bit manipulation, data input/output and simple data organization;
2. Consumer Devices: it focuses on multimedia applications and exploits audio and video processing algorithms;
3. Office Automation: it consists of text manipulation algorithms to represent office machinery, such as printers;
4. Networking: it includes shortest path calculations, tree and table lookups and data input/output workloads;
5. Security: it consists of common algorithms for data encryption, decryption and hashing applications;
6. Telecommunications: it includes radio frequency analysis, voice encoding/decoding and checksum algorithms.

It provides a small data and large data set to represent two different levels of stress, but this requires access to a filesystem that may not always be available on small embedded platforms. It analyses the instruction distribution and compares three different architectures (simulated with SimpleScalar) with the *integer SPEC2000*, the current version of SPEC benchmark at the time of their writing, but does not analyze power consumption. Also, among the categories aforementioned, the closest one to represent our applications is Automotive and Industrial Control. However, its *basicmath* test performs only cubic function solving, integer square root and angle conversions from degrees to radians.

James Pallister et al. [60] developed the Bristol/Embecosm Energy Efficiency Benchmark Suite (BEEBS) with the purpose of exposing the energy consumption characteristics of microprocessors. It consists of programs chosen from available suites presented in Table 2.3 with their description and respective sources.

Name	Description
Blowfish (MiBench)	Encryption algorithm used in cryptography.
CRC32 (MiBench)	Verification of data streams often used to detect errors in data transmission.
Cubic root solver (MiBench)	Solves cubic equations.
Dijkstra (MiBench)	Shortest-algorithm path commonly used in network devices.
FDCT (WCET)	Finite discrete cosine transform used in many video decoders.
Float Matmult (WCET)	Floating-point matrix multiplication.
Integer Matmult (WCET)	Integer matrix multiplication.
Rijndael (MiBench)	Advanced Encryption Standard algorithm used in security applications.
SHA (MiBench)	Hashing algorithm used mainly for fingerprinting and data verification.
2D FIR (DSPstone)	Finite impulse response filter frequently used in image transformations.

Table 2.3: BEEBS benchmarks.

It collects the instruction distribution (integer, floating-point, memory, branch or other) per benchmark for different platforms. Through measuring the average power and instruction distribution per benchmark, linear regression is used to attribute an average power dissipation to each class of instructions.

Table 2.4 summarizes the informations discussed above. There is a certain lack in benchmarks focused on the embedded domain, mainly because the consortium concentrates all the main suppliers of microprocessors. However, recent works show that benchmark remains a relevant subject in both industry and academia.

	EEMBC	MiBench	BEEBS
Year	1997	2001	2016
Initial target	Embedded	Embedded	Compiler
Type	Small programs	Small programs	Small programs
Availability	Paid membership	Free	Free
Developer	Industry	Academia	Academia + Industry

Table 2.4: Embedded benchmarks description.

Two works are under development, MLPerf and EmBench, both with their 0.5 versions published at the time of this writing. MLPerf is aimed at evaluating machine learning hardware, software and services with respect to both training [61] and inference [62] performance. Training is the compute-intensive process where the neural network learns a new capability tuning weights based on massive datasets to achieve a quality metric. Inference utilizes the trained model to make useful predictions. Each benchmark is defined by a dataset, a quality target and for MLPerf Inference, a model and a latency constraint. It measures the time to either train a model or produce results using a trained model. The suites include workloads from image classification, object detection, translation, recommendation and reinforcement learning. Various other benchmarks have also emerged in the past from researches in the field of deep learning, such as DeepBench, DAWNbench, Fathom, Training Benchmarks for DNNs (TBD), AIMatrix, EEMBC MLMark, among others.

EmBench [63] is a trademark of the Embench Task Group of the Free and Open Source Silicon Foundation to develop a free and open-source benchmark suite of programs with no more than 64kB of ROM and 64kB of RAM. Its current version (EmBench 0.5) has 20 programs largely derived from BEEBS with branch, memory or integer computer intensive but no floating-point computation. The group intends to release the full version at the Embedded World Conference in February 2020.

2.2 The benchmark proposition

When selecting a benchmark it is important to choose one that has a set of instructions similar to the actual operation of the component in question. Although there are a variety of methods and metrics for measuring performance, few studies have been found about benchmarks concerned in measuring power consumption, especially for embedded systems. Moreover, none of the workloads aforementioned is suitable for ADCS applications running on an embedded microcontroller.

As stated in Section 2.1, the workload should explore floating-point operations of matrices and quaternions. The ULPMark suites are not freely available and MiBench suites in addition to not being directed to energy consumption, they use external files and do not have representative instructions. MLPerf and EmBench are still under development and also do not have floating-point calculations. The closest one is BEEBS, but among its programs, it only explores matrix multiplication, while there are other operations that are relevant to our purpose, for example, matrix inversion. Consequently, the proposal of a new benchmark is justifiable within this context.

2.2.1 Implementation

The workload proposed is based on floating-point calculations implemented in C language and consists of the following operations:

1. Addition of $m \times n$ matrices;
2. Multiplication of $m \times n$ and $n \times p$ matrices;
3. Inversion of a $n \times n$ matrix;
4. Transposition of a $n \times n$ matrix;
5. Adjoint of a $n \times n$ matrix;
6. Cofactor from ij^{th} position of a $n \times n$ matrix;
7. Cofactor matrix of a $n \times n$ matrix;
8. Quaternion products (defined in equations B.21 and B.22 on Appendix B);
9. Determinant of a $n \times n$ matrix;
10. Trace of a $n \times n$ matrix;
11. Norm of a n -dimensional vector.

These functions compose a C library used to implement the code on Algorithm 1. Each function is executed for a given number of iterations and execution time is obtained with a system call or with a timer for those that do not support operating systems.

```

Declare input and output variables;
for  $i = 0$  to number of iterations do
    Clear timer and/or variables;
    Start timer;
    for  $f$  in  $\mathcal{F}$  do
        | Execute function  $f$ 
    end
    End timer;
    Store difference between start and finish;
end
Calculate average time and standard deviation;

```

Algorithm 1: Benchmark pseudocode.

In this implementation, the input variables are two 3×3 matrices, A and B, and two quaternions, Q_A and Q_B and the set \mathcal{F} of operations is:

- | | |
|--|--|
| f_1 Addition of A and B; | f_8 Quaternion product $Q_A \odot Q_B$; |
| f_2 Multiplication of A and B; | f_9 Determinant of A; |
| f_3 Inversion of A; | f_{10} Determinant of B; |
| f_4 Inversion of B; | f_{11} Trace of A; |
| f_5 Transposition of A; | f_{12} Trace of B; |
| f_6 Transposition of B; | f_{13} Norm of Q_A ; |
| f_7 Quaternion product $Q_A \otimes Q_B$; | f_{14} Norm of Q_B . |

With respect to implementation targeted to embedded systems, each function receives its output variable avoiding dynamic memory allocation and the matrices are manipulated as a one-dimensional array. Three auxiliary arrays are also statically allocated within the library itself. The output variables are overwritten as each function is executed. These output variables are one array of $n \times p$ elements to compute matrix operations, a 1×4 vector for quaternion operations and a scalar for other operations.

2.2.2 Metrics and rules

Energy consumption is an important design constraint in most embedded systems, from small handheld devices to servers clusters. In addition, the EPS poses concerns for all space-crafts and can be attributed to 27% of mission failures [64]. In particular, CubeSats have

limited power available due to its constrained size and low efficiency of solar panels technologies. A 3U CubeSat can reach up to 30W by using deployable solar panels with an efficiency of less than 33% [65, ?]. Although the success rate of CubeSat missions is increasing over time [4], the EPS can be attributed to 14% of mission failures for the first 100 launched Cubesats [66].

To evaluate both performance and power consumption, the metric chosen was energy, the product of power and execution time ($E = P \times T$), which weights both variables equally. Figure 2.1 introduces the generic method used to measure power consumption that consists of an ammeter in series and a voltmeter in parallel with the device being measured. The power is calculated according to $P = V \times I$.

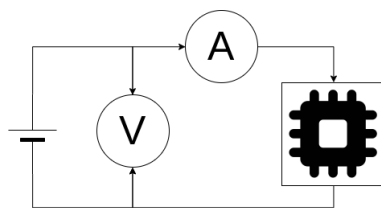


Figure 2.1: Generic setup of measurement.

The execution time is measured within the code and for fair comparisons, it is important that the compilers apply the same amount of optimization, otherwise, it may overshadow speed differences. In order to obtain the worst case scenario, the clock frequency should be set manually to its maximum value before running the benchmark.

This set-up allowed real measurements to be taken, rather than using an abstract power model. However, this adds systematic errors from instrumentation. Thus, the result should be obtained as a geometric average of several benchmark rounds, as well as the standard deviation of these measurements.

2.3 Target devices

This section exposes the real hardware platforms used for validating the benchmark. Firstly, a survey about the microprocessors used in commercial solutions was conducted, presented in Table 2.5.

Vendor	Product	Manufacturer	Processor
GomSpace	Nanomind A712D	ATMEL	ARM7TDMI
GomSpace	Nanomind A3200	ATMEL	AVR32
GomSpace	Nanomind Z7000	Xilinx	ARM Cortex-A9 + FPGA
ISIS	ISIS OBC	ATMEL	ARM 9
GAUSS Srl	Abacus	Texas Instruments	MSP430 + FPGA
GAUSS Srl	Hercules	Texas Instruments	ARM Cortex R4F
Pumpkin	PSPM D/E, PPM D1/D2	Microchip	PIC
Pumpkin	PPM A1/A2/A3, FM430	Texas Instruments	MSP430
NanoSatisfi	ArduSat	ATMEL	AVR

Table 2.5: Commercial onboard computers microprocessors (adapted from [?, ?]).

The ARM, PIC, MSP and FPGAs families are widely used in this market, therefore, a set of representative development boards are chosen as follows: Texas Instrument MSP430FR5994 Launchpad Kit, Arduino Uno Rev.3, ST Nucleo-L432KC and Raspberry PI 3B. Table 2.6 displays their hardware specifications.

Specification	Arduino	MSP430	Nucleo L4	Raspberry Pi 3B
Processor	Atmega328P-PU	MSP430FR5994	STM32L4KC	Broadcom BCM2837
Architecture	AVR	MSP430	ARM Cortex M4	4 x ARM Cortex A53
Bit width	8-bit	16-bit	32-bit	64-bit
Frequency	20MHz	16MHz	80Mhz	1.2GHz
Temperature	-40°C to +85°C	-40°C to +85°C	-40°C to +85°C	-10°C to +50 °C
Supply Voltage	1.8V to 5.5V	1.7V to 3.6V	1.71V to 3.6V	2.5V to 5.5V
Floating-point unit	No	MPY	Yes	Yes
Memory	Flash 32kB	FRAM 256kB	Flash 256kB	RAM 1GB
	EEPROM 1kB RAM 2kB	SRAM 8kB	SRAM 64kB	

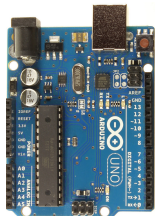








Table 2.6: Hardware specifications.

It is noteworthy that Raspberry Pi is a different category from the other microcontrollers studied. It has a much higher processing capacity since it is quad-core, 64-bit, with a much higher frequency than the previous ones and many more available peripherals, such as WiFi and Ethernet module, audio and HDMI outputs, GPU among others. However, it provides an interesting metric to compare such processing power within its constrained size.

2.3.1 Measurement procedures

In this section, the methodologies and considerations adopted for measuring power and time consumption for each hardware are explained.

2.3.1.1 Measuring power consumption

In Section 2.2.2 it was described a generic methodology to measure power consumption by using an ammeter and a voltmeter. For our measurements, it was used two Minipa ET-1110 DMM, one for measuring voltage and other for current, as it provides multiple scales for current measurements (μA and mA). The Raspberry Pi 3B, Nucleo-L432KC and Arduino Uno boards were powered at 5V, while MSP430FR5994 Launchpad kit was powered at 3.3V with a DC power supply (EQ030F). Figure 2.2 illustrates this setup, where the Raspberry Pi was used as a representation.

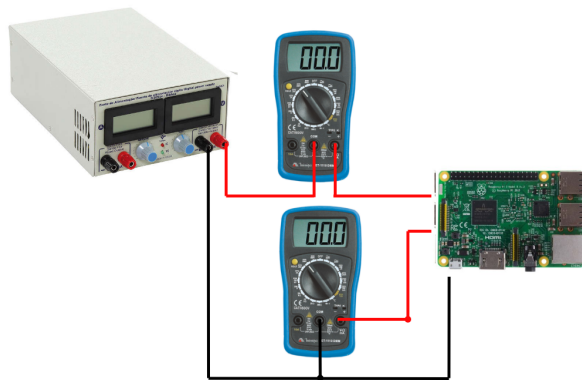


Figure 2.2: Setup of measurement.

2.3.1.2 Measuring execution time

In this section, it will be explained the configurations implemented to measure the time, as well as the software used during programming and debugging of each hardware.

Arduino Uno - The ATmega328P has two 8-bit timers and one 16-bit timer that was configured to measure execution time. Using timers yielded in more precise results than using the *micros()* or *millis()* functions of the standard Arduino library. Although the datasheet states that the maximum frequency is 20MHz at 5V, the standard Arduino frequency is 16MHz to ensure compatibility with older microprocessor models.

The Arduino code was compiled with `avr-gcc` on Arduino IDE v. 1.8.10. It is important to highlight that it does not have a floating-point unit or a dedicated hardware multiplier.

MSP430FR5994 Launchpad kit - The chip in question, MSP430FR5994, has different timer instances that can be configured in multiple modes. Timer TA2 has been set to measure the time interval in continuous mode. This timer is sourced from the sub-main clock

(SMCLK) at 1MHz.

Some devices of the MSP430 family have a peripheral module called the hardware multiplier. In our case, there is a 32-bit hardware multiplier that supports 8-bit, 16-bit, 24-bit and 32-bit operands. It supports unsigned or signed multiplications with or without accumulate, as well as fractional numbers.

The Texas Instruments Code Composer Studio v. 9.1.0.10 was used to program and debug the MSP launch kit, with TI compiler v18.12.2.LTS.

Nucleo-L432KC - Some ARM architectures, such as Cortex M4, have a Data Watchpoint and Trace (DWT) unit implemented, which contains up to four comparators that can be configured as desired. It presents a counter of clock cycles (CYCCNT) that was used in the code to measure execution time. The device also features a floating-point unit (FPU) single precision.

For programming and debugging the Nucleo-L432KC board, it was used the software STMicroelectronics System Workbench v. 4.6.3, an Eclipse based IDE, that uses compiler GCC Cross Compiler v. 9.2.1.201704050430.

STMicroelectronics has an interesting software for hardware configurations, the STM32CubeMX v. 5.3.0, where it is possible to manage peripherals and set the clock source and frequency, that was configured at 80MHz.

Raspberry Pi 3B - Raspberry is the only one that has its own operating system. It is based on Debian Buster, the Raspbian v. 4.19. The code was compiled with GNU Compiler v. 9.1. Execution time was measured with a system call `clock_gettime()` using a single core with a fixed frequency of 1.2GHz. This particular configuration was achieved by modifying two system archives:

- `/boot/cmdline.txt`: added `maxcore=1` (reboot needed)
- `cpu0/cpufreq/scaling_governor`: changed `ondemand` to `performance`

2.4 Summary

This chapter presented the specification and implementation of a complete benchmark concerned at evaluating the energy consumption of elementary operations from nanosatellites control system software. The proposed benchmark can be summarized as the following three main specifications:

- **Workload**: comprises a set of floating-point operations on matrices and quaternions written in C language. The proposed set is defined by 14 operations listed in Section 2.2.1;

- Metric: the proposed metric weighs performance and power equally and uses energy consumption as the output, that is, the product of execution time and power;
- Rules: for running the workload, it is important that the compilers apply the same amount of optimization, and the clock frequency should be configured to its maximum value. For measuring energy consumption, the execution time should be calculated within the algorithm itself (see Algorithm 1: Benchmark pseudocode) and the current and voltage should be measured with appropriate external instruments, such as ammeter and voltmeter. In both cases, in order to obtain the best value (and thus improve accuracy and precision), it is suggested to perform measurements under repeatability conditions using the same instruments and under the same environmental conditions to obtain a mean value. This average value is expected to become more accurate as the number of repetitions increases. The uncertainty in a specific result will be specified by the standard deviation, that is, the standard uncertainty.

The chapter also presented the hardware and software specifications of the platforms used for validating the benchmark and the measurement procedures for each of them.

Chapter 3

Experimental results

This chapter presents the results obtained by running the benchmark on different micro-controllers. To get as close as possible to the application in question, the input matrices have dimensions 3×3 since they are representations of attitude. All the values have the same magnitude of the real variables, represented as a float data type with four significant digits, taken from the book [67]:

$$A = \begin{bmatrix} 0.3520 & 0.8640 & 0.3600 \\ -0.8640 & 0.1520 & 0.460 \\ 0.3600 & -0.4800 & 0.8000 \end{bmatrix} \quad B = \begin{bmatrix} -0.5736 & 0.0000 & -0.8192 \\ 0.0496 & 0.8660 & -0.2868 \\ 0.7094 & -0.5000 & -0.4967 \end{bmatrix}, \quad (3.1)$$

$$q_A = \begin{bmatrix} \frac{\sqrt{2}}{2} \\ 0.0000 \\ 0.0000 \\ \frac{\sqrt{2}}{2} \end{bmatrix} \quad q_B = \begin{bmatrix} 0.6853 \\ 0.6953 \\ 0.1531 \\ 0.1531 \end{bmatrix}. \quad (3.2)$$

CubeSats are mostly launched into Low Earth Orbit with altitudes ranging from 200 to 1000km. At this range, the space environment is extremely adverse and can affect the system operation in many different ways. Atomic oxygen (AO), ultraviolet (UV) radiation, ionizing radiation, ultra-high vacuum (UHV), thermal cycles and orbital debris can cause deterioration of materials and electronic components. This type of mission is tolerable to certain risks compared to traditional spacecraft, but even so, the pre-launch process must follow some testing and integration phases to prove the safety and robustness of the CubeSat [68].

The testing phase typically includes vibration and thermal vacuum tests and it is known that these variables may cause malfunction even within the operating range provided by the suppliers. The following tests were performed under ambient conditions (32°C and 900hPa) and do not analyze the behavior of the microcontrollers for variations of these conditions.

The benchmark was validated using the following development platforms: Arduino Uno,

MSP430FR5994 Launchpad Kit, Nucleo L432KC and Raspberry Pi 3B. At this point, it is interesting to note that the number of iterations of each round was limited by the Arduino memory size, which corresponds to 700 iterations. Table 3.1 shows the code size for each platform.

Platform	Code size (kB)
Arduino Uno	9.252
MSP430FR5994	19.116
Nucleo L432KC	33.288
Raspberry Pi 3B	13.564

Table 3.1: Code size.

The above results include both code and data values. The Arduino code is only 9kB, but more than 1.8kB is used for global variables that are stored in RAM, which represents 91% of the total space (as shown in Table 2.6, Arduino has only 2kB of RAM) thus limiting the number of iterations per round. The Nucleo L432KC code has the greatest size because the *System Workbench* has a hardware abstraction layer (HAL) that is already incorporated into the project for hardware configurations, followed by MSP430FR5994 which also incorporates all libraries used in the implementation for clock configuration, timers and ports.

3.1 Execution time

The benchmark execution time is presented in Table 3.2. The results are the geometric mean and standard deviation for a single run with a number of iterations equal to 700.

Platform	Clock frequency (MHz)	Execution time (ms)
Arduino Uno	16	8.65 ± 0.0030
MSP430FR5994	16	12.24 ± 0.0005
Nucleo L432KC	80	$0.94 \pm + 0.0002$
Raspberry Pi 3B	1200	$0.05 \pm + 0.0064$

Table 3.2: Elapsed time.

Raspberry Pi results are the ones that vary the most for consecutive rounds, which is justified by the presence of its operating system that must perform tasks with higher priorities. In other cases, for consecutive rounds the results are almost always the same. Arduino Uno and MSP430 Launchpad have similar values as they have the same clock frequency, but the latter is much more accurate given the standard deviation obtained. The others, Raspberry and Nucleo board, are faster due to their higher frequency and the presence of a dedicated floating-point unit that improves their performance.

3.2 Energy consumption

The instrumentation for measuring energy consumption was explained in Section 2.3.1.1 and the results are presented in tables 3.3 and 3.4. The results presented are the geometric mean and standard deviation from 100 benchmark measurements.

Platform	Current (mA)	Voltage (V)	Power(mW)
Arduino Uno	33.01 ± 0.79	4.92 ± 0.03	162.52
MSP430FR5994	1.92 ± 0.11	3.39 ± 0.01	6.51
Nucleo L432KC	10.34 ± 0.72	4.90 ± 0.04	50.64
Raspberry Pi 3B	290.24 ± 18.82	4.90 ± 0.01	1422.01

Table 3.3: Power consumption.

From the results, it is seen that the Raspberry Pi is the most power consuming device among those presented. This result was expected by the number of peripherals it has (WiFi, HDMI, audio output, Ethernet) that were not turned off. The second largest in consumption was Arduino Uno, which can be explained by the presence of a second microcontroller (ATMEGA16U2) as a programming interface that consumes additional power. Next, is the Nucleo board followed by the MSP430 Launchpad, the latter being the only one to be powered at 3.3V. Figure 3.1 shows the power consumption per MHz, as the clock frequency is directly proportional to the power consumed.

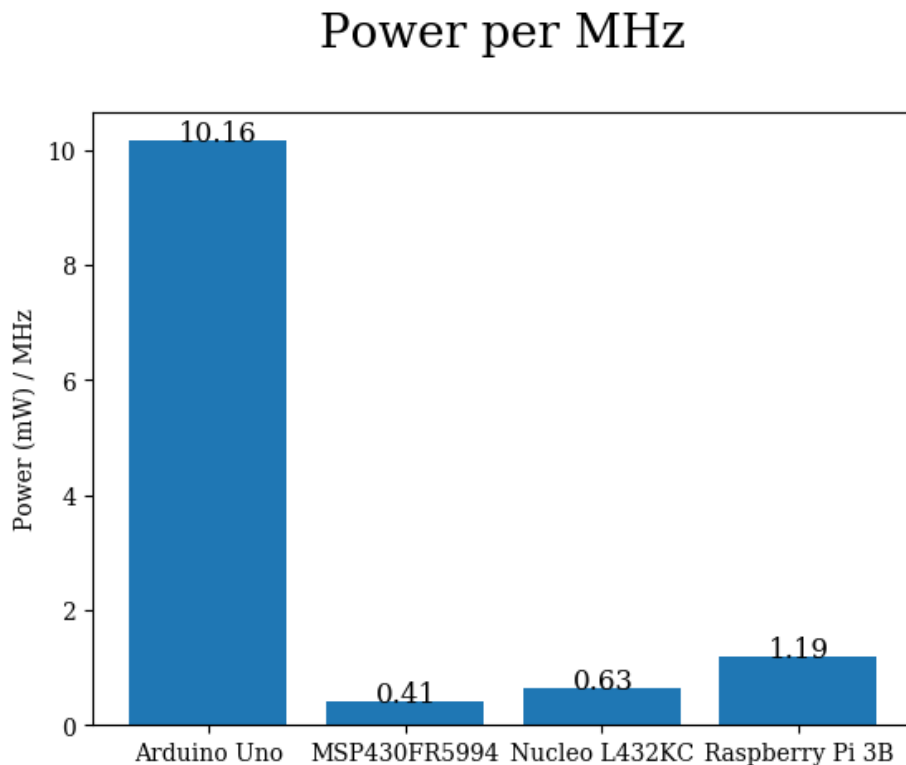


Figure 3.1: Power consumption per MHz (mW/MHz).

When we relate to their respective frequencies, Raspberry consumption offers an interesting trade-off in terms of its processing power, consuming much less than Arduino. Nucleo L432KC has a dedicated floating-point hardware, which shortens processing time but increases power consumption. MSP430 Launchpad kit is the platform that consumes less power per MHz.

Table 3.4 and Figure 3.2 present the results of energy consumption, which is the product of power by execution time.

Platform	Frequency (MHz)	Time (ms)	Power (mW)	Energy (mJ)
Arduino Uno	16	8.65	162.52	1.41
MSP430FR5994	16	12.24	6.51	0.08
Nucleo L432KC	80	0.94	50.64	0.05
Raspberry Pi 3B	1200	0.05	1422.01	0.07

Table 3.4: Power and energy consumption.

Energy consumption

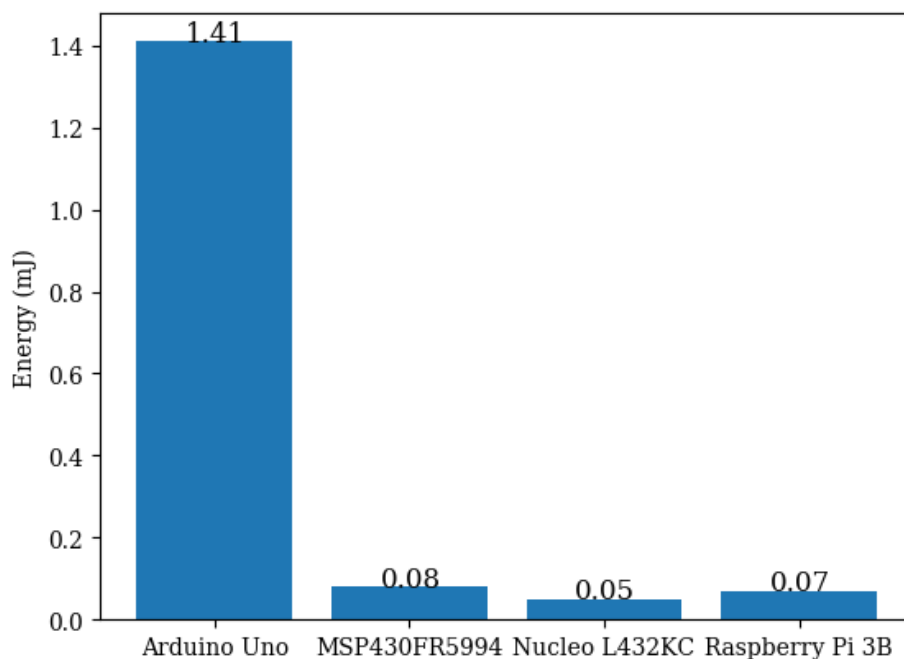


Figure 3.2: Energy consumed (mJ).

The Arduino Uno is the least energy-efficient. It consumes a lot of power and execution time to perform the same tasks. MSP430 Launchpad consumes almost the same energy as the Nucleo board, even consuming almost 10x less power. In addition to having a higher bit width, Nucleo board frequency is 5 times higher, reducing execution time. The MSP430 Launchpad consumes less power, but it takes longer, so energy consumption was higher.

As Raspberry Pi performs operations very fast due to its high clock speed, its bit-width and the presence of a FPU, the energy consumption ends up remaining in the same order of magnitude as the other devices evaluated, even though their power consumption is at least 8x higher.

An interesting point to note is that the Nucleo L432KC, even with a voltage regulator from 5V to 3.3V, has comparable energy consumption to MSP430FR5994 Launchpad, which was powered directly at 3.3V. Each function was evaluated separately inside a *while(1)* loop and the values for power consumption per MHz are presented in Figure 3.3.

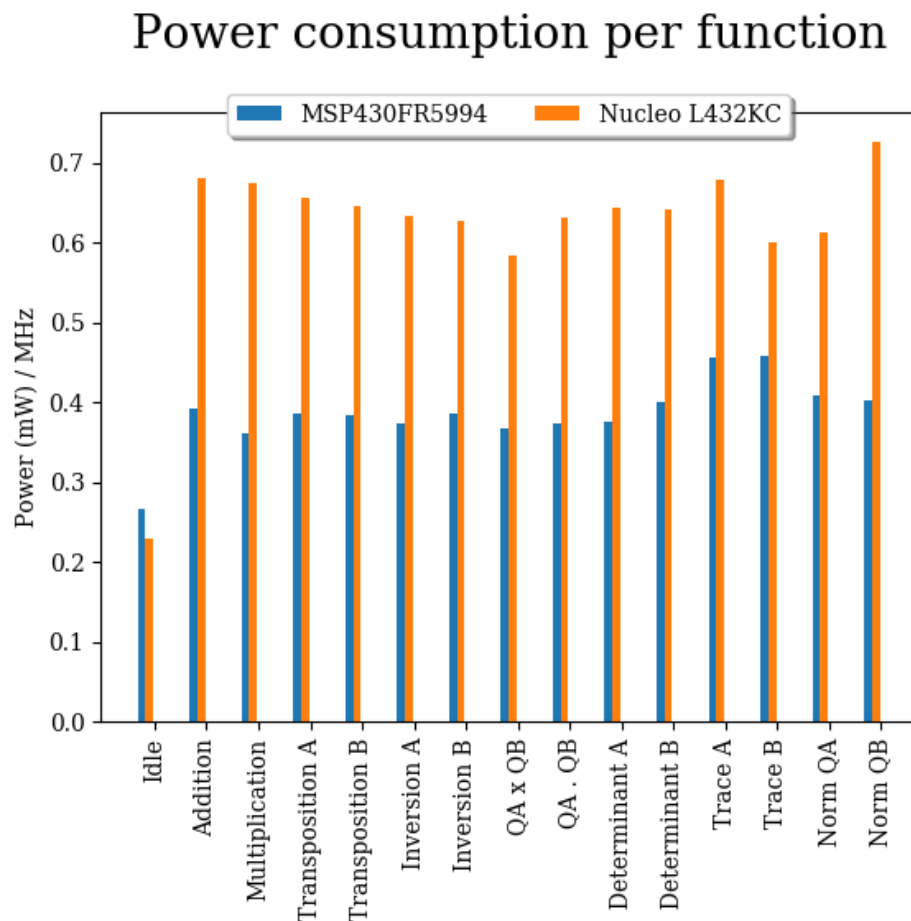
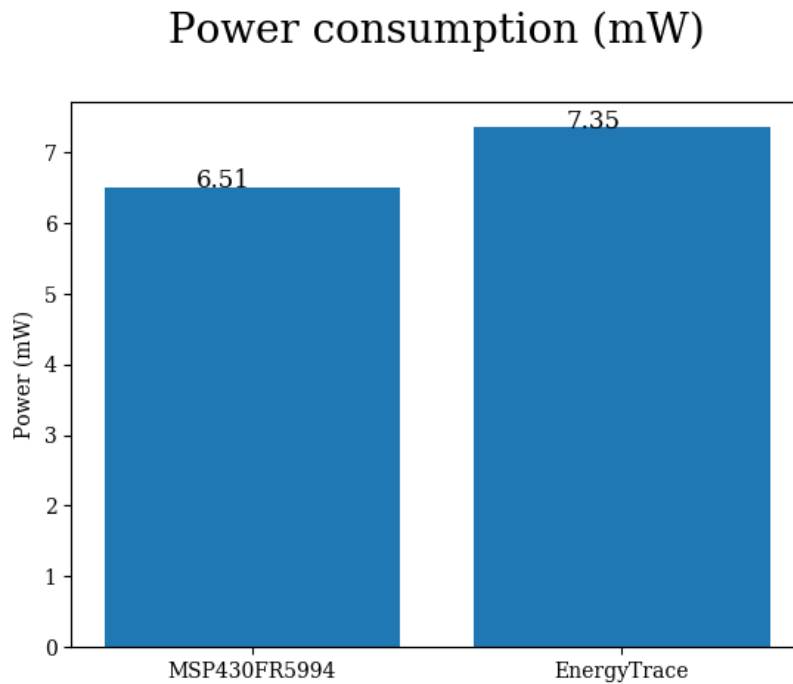


Figure 3.3: Power consumption (mW/MHz) per function.

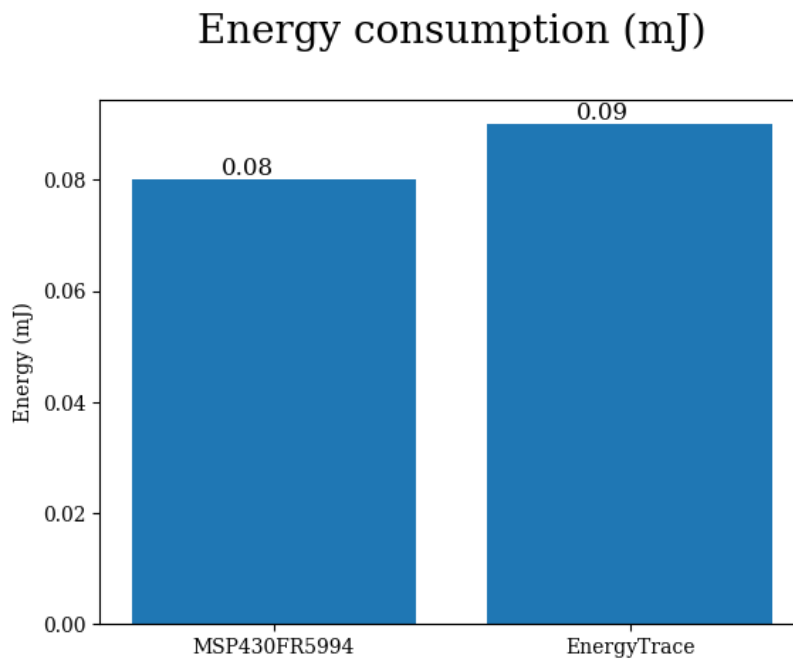
The results are the geometric average of 10 consecutive readings. Matrix multiplication represents the lowest consumption for the MSP430, while matrix B transposition the greatest. For Nucleo L4, the greater consumption is to calculate the norm of Q_B and the smaller is for calculating quaternion product $Q_A \otimes Q_B$.

3.3 Reference platform

The MSP4305994 Launchpad was chosen as a reference platform to relate the results because it has a built-in tool that allows real-time energy monitoring, called *EnergyTrace++*. Figure 3.4 presents a comparison of the consumption measured with this tool and the data previously presented.



(a) Power (mW)



(b) Energy (mJ)

Figure 3.4: EnergyTrace++ comparison

We have a difference of 11.4% for both power and energy results that may come from the components used in the debugger part for EnergyTrace features but which is isolated during manual measurements. Having the exact same error for both quantities measured in different ways suggests that the results obtained are coherent.

Table 3.5 presents the time, power and energy ratios to the reference platform. Values less than 1 indicate that performance was better, as in the case of execution time in Raspberry Pi or Nucleo board time and energy.

Platform	Time	Power	Energy
Arduino Uno	0.707	24.956	17.636
MSP430FR5994	1.000	1.000	1.000
Nucleo L432KC	0.078	7.776	0.603
Raspberry Pi 3B	0.004	218.354	0.894

Table 3.5: Results related to a reference platform

3.4 LODESTAR applications

Finally, the results obtained are analyzed in the context of the workplace framework, i.e., LAICAnSat missions and air-bearing table simulations. As stated throughout this work, using a standardized platform like a CubeSat imposes three major design constraints: size, weight and power.

Figure 3.2 presents the energy consumed, which is directly related to the power capacity of batteries. That means that the MSP430 family requires more energy (bigger electric charge) which directly impacts the size of the set of batteries, consequently increasing the total mass of the system. However, we can see in Figure 3.1 and Table 3.3 that the MSP430 requires less current than the others. That suggests a trade-off between how much energy is stored in the battery and the amount of current the battery draws for running the same application.

Futhermore, others requirements must be taken into account, such as communication interfaces available, analog-to-digital converters and timers. Since the Raspberry Pi does not have low-power modes, it is not interesting to embed it on atmospheric missions, but in the electronic system responsible for the simulator it may be the best option for its processing capability. MSP430FR5994 and STM32L4KC both seem useful solutions for the OBC. Further informations about the features available on their hardwares are presented in Table 3.6.

Peripherals	MSP430FR5994	STM32L4KC
LPM	5	7
GPIOS	40	26
DMA	6-channel	14-channel
ADC	20-channel	10-channel
Timers	6	8
		2 x I2C
		2 x SPI
		2 x USART
Communication	2 x I2C	1 x LPUART
interfaces	3 x SPI	1 x SAI
	2 x UART	1 x CAN
		1 x USB
		1 x SWPMI

Table 3.6: Hardware peripherals.

Once more, both have similarities. However, it was shown that STM32 has greater processing capabilities than MSP430 for very close power usage. It also possesses more communication interfaces and timers, which are responsible for generating PWM outputs needed to control actuators. In addition, it is capable of more data handling for it has a larger direct memory access (DMA) and it has two more low-power modes (LPM) than MSP. Therefore, STM32 is more suitable to attend LAICAnSat demands.

Chapter 4

Conclusion

The context of this work was the elaboration of a new benchmark motivated by the onboard control system software of nanosatellites. The proposal of a new workload was justified by the absence of a set of elementary instructions relevant to evaluate both performance and power usage for ADCS applications running on commercial microcontrollers for a nanosatellite OBC. After the analysis of attitude determination and control methods currently used in the segment, the implementation followed a set of floating-point operations of 3×3 matrices and quaternions (1×4 vectors). The workload covers various fields of controllers and filter designs besides this particular application, as stated in Section 2.1.

The benchmark selected energy usage as a metric to compare power consumption and performance on different hardware configurations and the operational rules for running and measuring it were presented. The implementation was validated by running it on development platforms with microcontrollers from within common families of commercial onboard computers, namely Arduino Uno, MSP430FR5994 Launchpad, Nucleo L432KC and Raspberry Pi 3B.

It was shown that MSP430FR5994 consumes less power than all evaluated devices, while Raspberry Pi consumes the most. However, energy consumption was higher for Arduino Uno and lower for Nucleo L432KC. Raspberry Pi 3B provided an interesting result given the difference in its processing capabilities from the other ones. It is more power-efficient than Arduino even with more built-in features, for instance, WiFi, HDMI and Ethernet modules besides supporting an operating system.

Finally, these results were also analyzed with the purpose of using these platforms in the projects currently developed at the LODESTAR, where the Raspberry Pi was chosen for the applications of the nanosatellite simulator facility and the Nucleo L432KC for LAICAnSat activities.

4.1 Future works

- In Section 2.1.1, the benchmarks designed for embedded systems usually consist of a suite with more than one program to evaluate different features of a particular application. This benchmark considered only attitude determination and control algorithms, therefore it performs mathematically intense calculations. An improvement would be to add functions that are representative of other subsystems. For instance, command and data handling functions, such as bit manipulation and data organization, security functions, such as error detection, encryption/decryption algorithms, among others.
- It would be interesting to have a larger database for future design choices. For that, it is necessary to add support for devices other than the four that were compared.
- A widely used device in space applications is the FPGAs. These can greatly improve the performance of matrix operations and porting the code to their language (hardware description language) would also be interesting.

REFERENCES

- [1] MABROUK, E. *What are SmallSats and CubeSats?* 2015. Available at: <<https://www.nasa.gov/content/what-are-small-sats-and-cubesats>>. Accessed on: 02 sep. 2019.
- [2] HEIDT, H. et al. CubeSat: A new generation of picosatellite for education and industry low-cost space experimentation. *AIAA-USU Small Satellite Conference Proceedings*, v. 5, 2000.
- [3] CENTRO DE GESTÃO E ESTUDOS ESTRATÉGICOS - CGEE. Cubesats. Brasília – DF: 2018. 46p.
- [4] VILLELA, T. et al. Towards the thousandth CubeSat: a statistical overview. *International Journal of Aerospace Engineering*, Hindawi, 2019.
- [5] CALIFORNIA POLYTECHNIC STATE UNIVERSITY. *CubeSat Design Specification*. Rev. 13, 2015.
- [6] LUMBWE, L. T. Development of an onboard computer (OBC) for a CubeSat. Cape Peninsula University of Technology, 2013. PhD dissertation.
- [7] BORSCHIOV, K. Generic on-board-computer hardware and software development for nanosatellite applications. York University, 2012. Master thesis.
- [8] WERTZ, J. R. *Spacecraft attitude determination and control*. Springer Science & Business Media, 2012.
- [9] RAZZAGHI, E. Design and qualification of on-board computer for Aalto-1 CubeSat. Aalto University, 2012. Master thesis.
- [10] WERTZ, J. R.; EVERETT, D. F.; PUSCHELL, J. J. *Space mission engineering: the new SMAD*. Microcosm Press, 2011.
- [11] AMADO, R. C. Determinação de sistema de reentrada com paraquedas da plataforma LAICAnSat. Universidade de Brasília, 2018. Undergraduate thesis.
- [12] HONDA, Y. H. M. Análise e controle da trajetória do LAICAnSat-3. Universidade de Brasília, 2017. Undergraduate thesis.

- [13] NEHME, P. H. D. et al. Development of a meteorology and remote sensing experimental platform: The LAICAnSat-1. In: IEEE. *2014 IEEE Aerospace Conference*, 2014. p. 1–7.
- [14] ALVES, M. F. S. et al. Design of the structure and reentry system for the LAICAnSat-3 platform. In: *II Latin American Cubesat Workshop*, 2016. p. 1–15.
- [15] HOLANDA, M. A. L. et al. Trajectory control system for the LAICAnSat-3 mission. In: IEEE. *2017 IEEE Aerospace Conference*, 2017. p. 1–7.
- [16] NORONHA, B. H. A. et al. System identification of a square parachute and payload for the LAICAnSat. In: IEEE. *2015 IEEE Aerospace Conference*, 2015. p. 1–7.
- [17] DIAS, R. R. et al. LAICAnSat-3: A mission for testing a new electronic and electronic and telemetry and tracking system. In: *Proceedings of the 2nd Latin American IAA Cubesat Workshop*, 2016. p. 1–9.
- [18] BORGES, R. A. et al. LAICAnSat-5: A mission for recording the total solar eclipse from the stratosphere. In: IEEE. *2018 IEEE Aerospace Conference*, 2018. p. 1–7.
- [19] HONDA, Y. H. M. Estudo de um sistema de controle de altitude para plataformas atmosféricas. Universidade de Brasília, 2019. Master thesis.
- [20] BARBOSA, V. H. C. Balloonsats: Projeto de missão e de um sistema de recuperação de carga paga. Universidade de Brasília, 2019. Undergraduate thesis.
- [21] BARBOSA, V. C. et al. Development of an actuator for an airdropped platform landing system. In: *Proceedings of the 41st IEEE Aerospace Conference*, 2020. p. 1–7.
- [22] PLOEG, L. C. van der. Desenvolvimento de sistema para simulação do campo magnético terrestre em órbitas baixas. Universidade de Brasília, 2017. Undergraduate thesis.
- [23] SILVA, R. C. da. Filtering and adaptive control for balancing a nanosatellite testbed. Universidade de Brasília, 2018. Master thesis.
- [24] GUIMARÃES, F. C. Implementation of attitude determination techniques for a small satellite three-axis simulator. Universidade de Brasília, 2018. Master thesis.
- [25] SILVA, R. C. da et al. Helmholtz cage design and validation for nanosatellites HWIL testing. *IEEE Transactions on Aerospace and Electronic Systems*, IEEE, 2019.
- [26] SILVA, R. C. da et al. Tabletop testbed for attitude determination and control of nanosatellites. *Journal of Aerospace Engineering*, American Society of Civil Engineers, v. 32, n. 1, p. 04018122, 2018.
- [27] LOIOLA, J. V. L. et al. Development of a hardware-in-the-loop test platform for nanosatellites ADCS integrated with an UKF. In: *4th Conference on University Satellite Missions and Cubesat Workshop*, 2017. p. 365–373.

- [28] SILVA, R. C. et al. A testbed for attitude and determination control of spacecrafts. In: *II IAA Latin American Cubesat Workshop*, 2015.
- [29] ISHIOKA, I. S. K. et al. HIL testing of the B-dot attitude control law. In: IAA. *III IAA Latin American Cubesat Workshop*, 2018. p. 1–9.
- [30] ISHIOKA, I. S. K. et al. Development of an active magnetic actuator for attitude control system of nanosatellites. In: *4th Conference on University Satellite Missions and Cubesat Workshop*, 2017. p. 327–342.
- [31] LOIOLA, J. V. L. de et al. 3 axis simulator of the Earth magnetic field. In: IEEE. *2018 IEEE Aerospace Conference*, 2018. p. 1–8.
- [32] SHAO, A.; WALKER, A. State of the art: benchmarking microprocessors for embedded automotive applications. *International Journal of Advanced Computer Research*, International Journal of Advanced Computer Research, v. 6, n. 26, p. 185, 2016.
- [33] RIVOIRE, S. et al. JouleSort: a balanced energy-efficiency benchmark. In: ACM. *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, 2007. p. 365–376.
- [34] CRASSIDIS, J. L.; MARKLEY, F. L.; CHENG, Y. Survey of nonlinear attitude estimation methods. *Journal of Guidance, Control, and Dynamics*, v. 30, n. 1, p. 12–28, 2007.
- [35] van der Ha, J. C.; Shuster, M. D. A tutorial on vectors and attitude. *IEEE Control Systems Magazine*, v. 29, n. 2, p. 94–107, April 2009.
- [36] MARKLEY, F. L.; CRASSIDIS, J. L. *Fundamentals of Spacecraft Attitude Determination and Control*. Princeton University Press, 2014.
- [37] WERTZ, J. *Spacecraft Attitude Determination and Control*. Springer Netherlands, 2002. (Astrophysics and Space Science Library).
- [38] YANG, Y. *Spacecraft Modeling, Attitude Determination, and Control: Quaternion-Based Approach*. Boca Raton: CRC Press, 2019.
- [39] WON, C.-H. Comparative study of various control methods for attitude control of a leo satellite. *Aerospace Science and Technology*, v. 3, n. 5, p. 323 – 333, 1999.
- [40] Chaturvedi, N. A.; Sanyal, A. K.; McClamroch, N. H. Rigid-body attitude control. *IEEE Control Systems Magazine*, v. 31, n. 3, p. 30–51, June 2011.
- [41] Raitoharju, M.; Piché, R. On computational complexity reduction methods for Kalman filter extensions. *IEEE Aerospace and Electronic Systems Magazine*, v. 34, n. 10, p. 2–19, Oct 2019.

- [42] ASTRÖM, K. J.; HÄGGLUND, T. *PID Controllers: Theory, Design, and Tuning*. 2nd ed. ISA: The Instrumentation, Systems, and Automation Society, 1995.
- [43] RAZ, R. On the complexity of matrix product. *SIAM Journal on Computing*, v. 32, n. 5, p. 1356–1369, 2003.
- [44] WIGDERSON, A. *Mathematics and Computation: A Theory Revolutionizing Technology and Science*. Princeton University Press, 2019.
- [45] BLONDEL, V. D.; TSITSIKLIS, J. N. A survey of computational complexity results in systems and control. *Automatica*, v. 36, n. 9, p. 1249–1274, 2000.
- [46] KUSHEL, O. Y. Unifying matrix stability concepts with a view to applications. *SIAM Review*, v. 61, n. 4, p. 643–729, 2019.
- [47] HADDAD, W. M.; CHELLABOIN, V. S. *Nonlinear Dynamical Systems and Control: A Lyapunov-Based Approach*. Springer, New York, NY, 2011.
- [48] WHETSTONE (benchmark). In: WIKIPÉDIA: a enciclopédia livre. Wikimedia, 2019. Available at: <[https://en.wikipedia.org/wiki/Whetstone_\(benchmark\)](https://en.wikipedia.org/wiki/Whetstone_(benchmark))>. Accessed on: 06 jan. 2019.
- [49] CURNOW, H. J.; WICHMANN, B. A. A synthetic benchmark. *The Computer Journal*, Oxford University Press, v. 19, n. 1, p. 43–49, 1976.
- [50] WHETSTONE benchmark history and results. In: ROY Longbottom’s PC Benchmark Collection. Roy Longbottom, 2014. Available at: <<http://www.roylongbottom.org.uk/whetstone.htm>>. Accessed on: 06 jan. 2019.
- [51] WEICKER, R. P. An overview of common benchmarks. *Computer*, IEEE, v. 23, n. 12, p. 65–75, 1990.
- [52] DHRYSTONE benchmark results on PCs. In: ROY Longbottom’s PC Benchmark Collection. Roy Longbottom, 2017. Available at: <<http://www.roylongbottom.org.uk/dhrystone%20results.htm>>. Accessed on: 06 jan. 2019.
- [53] DONGARRA, J. J. The LINPACK benchmark: An explanation. In: SPRINGER. *International Conference on Supercomputing*, 1987. p. 456–474.
- [54] STANDARD Performance Evaluation Corporation, 2019. Available at: <<https://www.spec.org/>>. Accessed on: 02 sep. 2019).
- [55] CORPORATION, S. P. E. *SPEC Power*®. 2019. Available at: <https://www.spec.org/power_ss_j2008/>. Accessed on: 06 jan. 2019.
- [56] WARRENTON, V. *SPEC releases power-performance benchmark for servers*. 2007. Available at: <https://www.spec.org/power_ss_j2008/press/SPECpower_ss_j2008-Press%20Release.html>. Accessed on: 06 jan. 2019.

- [57] WEISS, A. R. The standardization of embedded benchmarking: Pitfalls and opportunities. In: IEEE. *Proceedings 1999 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, 1999. p. 492–508.
- [58] POOVEY, J. A. et al. A benchmark characterization of the EEMBC benchmark suite. *IEEE micro*, IEEE, v. 29, n. 5, p. 18–29, 2009.
- [59] GUTHAUS, M. R. et al. MiBench: A free, commercially representative embedded benchmark suite. In: IEEE. *Proceedings of the Fourth Annual IEEE International Workshop on Workload Characterization. WWC-4 (Cat. No. 01EX538)*, 2001. p. 3–14.
- [60] PALLISTER, J.; HOLLIS, S. J.; BENNETT, J. *BEEBS: Open Benchmarks for Energy Measurements on Embedded Platforms*. Cornell University, 8 2013.
- [61] MATTSON, P. et al. Mlperf training benchmark. *arXiv preprint arXiv:1910.01500*, 2019.
- [62] REDDI, V. J. et al. Mlperf inference benchmark. *arXiv preprint arXiv:1911.02549*, 2019.
- [63] GROUP, E. T. *EmbenchTM user guide*. GitHub, 2019. Available: <<https://github.com/embench/embench-iot/blob/master/doc/README.md>>. Accessed on: 06 jan. 2019.
- [64] TAFAZOLI, M. A study of on-orbit spacecraft failures. *Acta Astronautica*, Elsevier, v. 64, n. 2-3, p. 195–205, 2009.
- [65] POGHOSYAN, A.; GOLKAR, A. CubeSat evolution: analyzing CubeSat capabilities for conducting science missions. *Progress in Aerospace Sciences*, Elsevier, v. 88, p. 59–83, 2017.
- [66] SWARTWOUT, M. The first one hundred CubeSats: A statistical look. *Journal of Small Satellites*, v. 2, n. 2, p. 213–233, 2013.
- [67] MARKLEY, F. L.; CRASSIDIS, J. L. *Fundamentals of spacecraft attitude determination and control*. Springer, 2014.
- [68] INITIATIVE, N. C. L. et al. Cubesat 101: Basic concepts and processes for first-time cubesat developers. *San Luis Obispo, USA*, 2017.

Appendix A

Reference frames

As seen in Section 2.1, the way a reference frame is defined can affect the observations relative to that frame. In general, a reference system is specified by its origin location and its axes orientation, the latter being the most important regarding to the study of attitude.

A reference frame consists of a set of three mutually orthogonal unit vectors that composes an orthonormal basis. An important aspect related to the definition of the frame is whether it is inertial or non-inertial. Inertial frames are non-accelerated frames, that is, those with respect to which the laws of Newton are valid. Non-inertial frames either translate or rotate with variable velocity and it introduces fictitious forces to the motion of the body.

A.1 Body-fixed frame

A frame fixed on the spacecraft is used to align the various hardware components and is extremely important for attitude determination. It is a non-inertial frame as it follows the rotation of the body, and its origin is usually placed at the center of the body, as seen in Figure A.1 for a 3U CubeSat. The frame is conveniently chosen according to the arrangement of the components within the body.

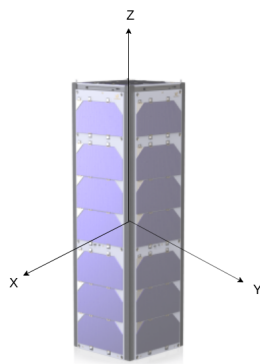


Figure A.1: 3U CubeSat body-fixed frame.

A.2 Local-Vertical/Local-Horizontal Frame

A common case of coordinate axes referenced to the spacecraft's orbit is the Local-Vertical/Local-Horizontal (LVLH), which follows the spacecraft in its orbit maintaining its orientation relative to the Earth. This representation is convenient for Earth-pointing applications. As shown in Figure A.2, the Z axis is placed towards the Earth's center of mass, the X axis is parallel to the orbital plane and has the same direction as the velocity vector for circular orbits. The Y axis points along the negative orbit normal, in the opposite direction to the spacecraft's orbital angular velocity.



Figure A.2: Local-Vertical/Local-Horizontal Frame.

A.3 Earth-Centered-Earth-Fixed

The Earth-Centered-Earth-Fixed (ECEF) is a non-inertial reference frame as it rotates with the Earth. Its origin is placed at the center of the planet, the X axis is the intersection between the equatorial plane and the prime meridian in Greenwich (0° latitude and 0° longitude). The Z axis is parallel to the spin axis of the Earth towards the North Pole, and Y axis is chosen applying right-handness property. Figure A.3 shows a representation of the ECEF reference system.

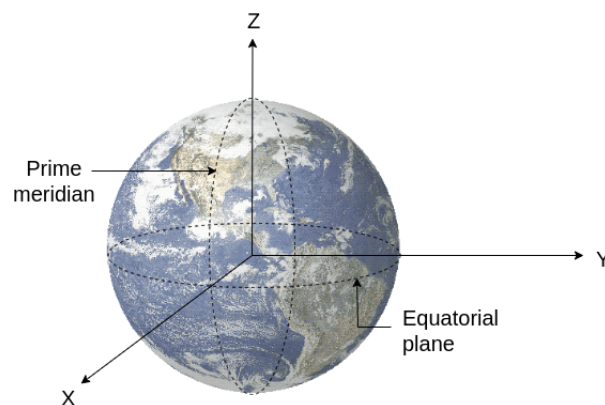


Figure A.3: ECEF reference frame.

A.4 Earth-Centered Inertial

The Earth-Centered Inertial (ECI) frame is useful for describing orbital motion of objects in space since it is a non-rotating frame and it is considered inertial relative to distant stars.

ECI origin point is fixed at the Earth's center of mass, with its X axis pointing towards the Sun at the vernal equinox¹, which is the intersection of the terrestrial equatorial plane with the ecliptic orbital plane. This point is slowly moving owing to the effects of astronomical precession and the nutation of the Earth's rotation axis, therefore, it has to be associated to a certain date, usually January 1st, 2000, as specified by J2000 standard.

The Z axis is parallel to the rotation axis of the Earth pointing to the North Pole and Y axis completes the right-handed coordinate frame. A representation of the ECEF system is exhibited in Figure A.4.

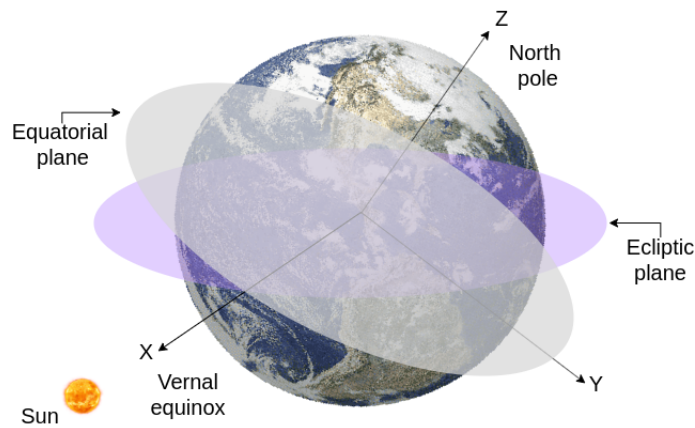


Figure A.4: ECI reference frame.

¹This direction points to Aries constellation, so it is commonly represented with Υ

Appendix B

Attitude Representations

The attitude of a body was defined as the rotation operation that takes the reference system into the body-fixed frame. This appendix presents some basic forms of mathematical representation for rotation matrices.

B.1 Definitions

For ease of understanding, the notations used throughout this chapter are described below:

- Vector are represented by bold minuscule letters: \mathbf{e}
- Matrices are represented by capital letters and their entries by the minuscule version of the same letter: first element of a matrix A is a_{11}
- On a three-dimensional space, a rotation may be seen as the transformation $\mathbf{a} = R_{ab}\mathbf{b}$, where R is a transformation matrix $R : \mathbf{a} \rightarrow \mathbf{b}$
- \mathbf{i} , \mathbf{j} and \mathbf{k} are unit vectors used to represent an orthonormal basis: $\mathbf{i} = [1, 0, 0]^T$, $\mathbf{j} = [0, 1, 0]^T$, $\mathbf{k} = [0, 0, 1]^T$
- This orthonormal base follows the the right hand rule: $\mathbf{i} \times \mathbf{j} = \mathbf{k}$, $\mathbf{j} \times \mathbf{k} = \mathbf{i}$, $\mathbf{k} \times \mathbf{i} = \mathbf{j}$
- Caresian axis X, Y and Z are provided as examples of arbitrary basis when needed

Rotation matrices belong to a particular group called proper real orthogonal matrices. This group comprises all the matrices of SO(3) group for rotation on a three-dimensional space. These 3×3 matrices have special features such as:

- Orthogonality: The unitary norm of vectors and the angle between them (90°) remain unchanged despite of a rotation:

$$\|\mathbf{i}\| = \|\mathbf{j}\| = \|\mathbf{k}\| = 1 \quad \mathbf{i}^T \mathbf{j} = \mathbf{i}^T \mathbf{k} = \mathbf{j}^T \mathbf{k} = 0$$

- Right-handedness property: Determinant must be equal to +1 in order to preserve the following:

$$\mathbf{i} \times \mathbf{j} = \mathbf{k}, \mathbf{j} \times \mathbf{k} = \mathbf{i}, \mathbf{k} \times \mathbf{i} = \mathbf{j}$$

- Its inverse equals its transpose:

$$R^{-1} = R^T$$

The parameterization of finite rotations of a rigid body was first described by Leonhard Euler (1707-1783), using three sequential rotations that resulted on a corollary of his theorem:

Theorem 1 *Any rotation can be described by a maximum of three successive elementary rotations about linearly independent axes*

This means any rotation matrix R can be represented as a decomposition of elemental rotations or arbitrary axes, provided successive rotations are not performed on collinear axes. Let X , Y and Z represent arbitrary axis:

$$R = R_X R_Y R_Z. \quad (\text{B.1})$$

Traditionally, it is designated an angle ϕ about X axis (*roll*), an angle θ about Y axis (*pitch*) and an angle ψ about Z axis (*yaw*), as shown in Figure B.1.

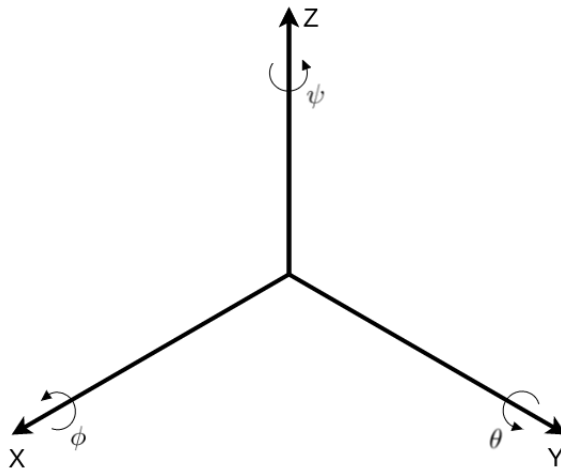


Figure B.1: Roll (ϕ), pitch (θ) and yaw (ψ) angles.

For rotations of such angles in these three axes we have the matrices B.2, B.3 and B.4. Note that each rotation matrix is a function of its axis representation, in other words, for

each of the elementary matrices, the row and columns associated with each axis preserve any vector along that axis unchanged.

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}, \quad (\text{B.2})$$

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}, \quad (\text{B.3})$$

$$R_z(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \quad (\text{B.4})$$

Therefore, given a known rotation matrix, $R(\phi, \theta, \psi)$, all three angles can be obtained by equations B.5 and B.6:

$$R(\phi, \theta, \psi) = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}. \quad (\text{B.5})$$

$$\phi = \text{atan2}(r_{32}, r_{33}) \quad \psi = \text{atan2}(r_{21}, r_{11}).$$

$$\theta = \begin{cases} \text{atan2}(-r_{31}, \frac{r_{21}}{\sin(\psi)}) & \text{if } \cos(\psi) = 0 \\ \text{atan2}(-r_{31}, \frac{r_{11}}{\cos(\psi)}) & \text{otherwise} \end{cases}. \quad (\text{B.6})$$

At this point, it is interesting to mention that the order of rotation is not commutative, as $R_X R_Y R_Z \neq R_Z R_Y R_X$. Such property is visualized proving that $AB \neq BA$:

$$AB \underbrace{(AB)^{-1}}_{=B^{-1}A^{-1}} = ABB^{-1}A^{-1} = AA^{-1} = I, \quad (\text{B.7})$$

$$BA(AB)^{-1} = BAB^{-1}A^{-1} \neq I. \quad (\text{B.8})$$

In this context, it is easy to see that there is a total of twelve possible combinations: six where all axes are different (e.g XYZ), known as *Cardan Angles*, and six where rotation occurs on the same axis (e.g XZX) known as *Euler Angles*.

From Equation B.1, the attitude matrix represented by the Euler angles ϕ , θ and ψ is

given by Equation B.9:

$$A_{ijk}(\phi, \theta, \psi) = A(e_k, \psi)A(e_j, \theta)A(e_i, \phi). \quad (\text{B.9})$$

Unfortunately, such a form of representation has singularities when the pitch angles $\theta = \pm\frac{\pi}{2}$. This is known as *gimbal lock*, when a degree of freedom is lost and angles cannot be determined uniquely. To work around this issue, one can either set the standard operating conditions away from or avoid maneuvers that pass through $\theta = \pm\frac{\pi}{2}$.

In the case of rotation applied to the attitude problem, the matrix is also known as direction-cosine matrix (DCM) by the way it is obtained as a relation between vectors from one reference frame and its transformation to another. Let $\mathbf{b} = [b_1 b_2 b_3]^T$ and $\mathbf{r} = [r_1 r_2 r_3]^T$ be the vector representations of body and reference frames, respectively, in a third coordinate frame, the attitude matrix A is defined as B.10

$$A = \begin{bmatrix} \langle b_1, r_1 \rangle & \langle b_1, r_2 \rangle & \langle b_1, r_3 \rangle \\ \langle b_2, r_1 \rangle & \langle b_2, r_2 \rangle & \langle b_2, r_3 \rangle \\ \langle b_3, r_1 \rangle & \langle b_3, r_2 \rangle & \langle b_3, r_3 \rangle \end{bmatrix}. \quad (\text{B.10})$$

where $\langle \mathbf{u}, \mathbf{v} \rangle$ is the inner product operation of two unit vectors, \mathbf{u} and \mathbf{v} , given by Equation B.11. Therefore, DCM is due to the fact each entry of A is the cosine of the angle α between two unit vectors.

$$\langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{u}^T \mathbf{v} = \cos(\alpha). \quad (\text{B.11})$$

Although DCM is the most natural way to represent attitude, it takes nine values to fully obtain the rotation matrix. In fact, there are only three (out of nine) independent parameters owing to the orthogonality constraint. There is also another corollary of Euler's Theorem that states an important properties of attitude matrices that serves for another option of representation:

Theorem 2 *Any combination of rotations on a rigid body can be described as a single rotation on fixed axis*

That means there is a single axis, typically represented by \mathbf{e} , known as an *Euler axis* in which the rotation operation is performed. This axis consists of the eigenvector associated with the eigenvalue equal to 1 ($A\mathbf{e} = \mathbf{e}$). Besides, it also is required one other parameter to completely obtain the rotation matrix: the angle of rotation, *Euler angle*, given by Equation B.12:

$$\cos \vartheta = \frac{\text{tr}(A) - 1}{2}. \quad (\text{B.12})$$

where $\text{tr}(A)$ denotes the trace of matrix A . The attitude matrix parameterized as Euler axis

and angle is given by Equation B.13:

$$A(\mathbf{e}, \vartheta) = I_3 - \sin(\vartheta)[\mathbf{e}\times] + (1 - \cos(\vartheta))[\mathbf{e}\times]^2. \quad (\text{B.13})$$

Sometimes it is convenient for analysis to combine the Euler axis and angles into a rotation vector, given by $\text{pmb}\vartheta = \vartheta\mathbf{e}$, which gives the attitude written as Equation B.14:

$$A(\mathbf{e}, \vartheta) = \exp([\vartheta\mathbf{e}\times]). \quad (\text{B.14})$$

where \exp is the matrix exponential. However, it is impractical for numerical computations due to its computational burden.

With an unit rotation vector plus the total rotation angle, it is possible to achieve a representation without singularities with fewer parameters than DCM. Equation B.15 presents an attitude representation based on these four parameters, known as Euler-Rodrigues symmetric parameters.

$$\begin{bmatrix} \eta_1 \\ \eta_2 \\ \eta_3 \\ \eta_4 \end{bmatrix} = \boldsymbol{\eta} = \hat{\mathbf{e}} \sin\left(\frac{\Phi}{2}\right), \eta_4 = \cos\left(\frac{\Phi}{2}\right). \quad (\text{B.15})$$

This four-component form of representation suggests using a more convenient form by making use of quaternions. Quaternions are a number system, conceived by Hamilton, which are extremely useful for describing rotations in three dimensions. A quaternion \mathbf{q} is a four-component vector that has a three-vector part and a scalar part. The position of the scalar part in the vector may differ depending on the notation used, so it is worth noting that in this manuscript, the same notation from [67], is adopted, which uses the scalar part as component q_4 : $\mathbf{q} = [\mathbf{q}_{1:3} \ q_4]^T$.

The attitude matrix with this parameterization is given by Equation B.16:

$$A(\mathbf{q}) = (q_4^2 - \|\mathbf{q}_{1:3}\|^2)I_3 - 2q_4[\mathbf{q}_{1:3}\times] + 2\mathbf{q}_{1:3}\mathbf{q}_{1:3}^T. \quad (\text{B.16})$$

A quaternion of unit norm can always be associated to a rotation in the same way a proper real orthogonal matrix can, so that it has analogous properties such as B.17, B.18:

$$\mathbf{q}^{-1} = \mathbf{q}^T, \quad (\text{B.17})$$

$$A(\mathbf{q})A(\hat{\mathbf{q}}) = A(\mathbf{q} \otimes \hat{\mathbf{q}}). \quad (\text{B.18})$$

It is possible to define a rotation quaternion with its components taken as the Euler sym-

metric parameters, as defined by Equation B.19:

$$\mathbf{q} = \begin{bmatrix} \mathbf{q}_{1:3} \\ q_4 \end{bmatrix} = \begin{bmatrix} \boldsymbol{\eta}_{1:3} \\ \eta_4 \end{bmatrix} \quad (\text{B.19})$$

Given two quaternions $\mathbf{q} = [\mathbf{q}_{1:3}q_4]^T$ and $\bar{\mathbf{q}} = [\bar{\mathbf{q}}_{1:3}\bar{q}_4]^T$, their operations are defined as follows:

- Addition:

$$\bar{\mathbf{q}} + \mathbf{q} = \begin{bmatrix} \bar{q}_1 + q_1 \\ \bar{q}_2 + q_2 \\ \bar{q}_3 + q_3 \\ \bar{q}_4 + q_4 \end{bmatrix}. \quad (\text{B.20})$$

- Dot product:

$$\bar{\mathbf{q}} \odot \mathbf{q} = \begin{bmatrix} q_4\bar{\mathbf{q}}_{1:3} + \bar{q}_4\mathbf{q}_{1:3} + \bar{\mathbf{q}}_{1:3} \times \mathbf{q}_{1:3} \\ \bar{q}_4q_4 - \bar{\mathbf{q}}_{1:3} \cdot \mathbf{q}_{1:3} \end{bmatrix}. \quad (\text{B.21})$$

- Cross product:

$$\bar{\mathbf{q}} \otimes \mathbf{q} = \begin{bmatrix} q_4\bar{\mathbf{q}}_{1:3} + \bar{q}_4\mathbf{q}_{1:3} - \bar{\mathbf{q}}_{1:3} \times \mathbf{q}_{1:3} \\ \bar{q}_4q_4 - \bar{\mathbf{q}}_{1:3} \cdot \mathbf{q}_{1:3} \end{bmatrix}. \quad (\text{B.22})$$

- Norm:

$$\|\mathbf{q}\| = \sqrt{q_1^2 + q_2^2 + q_3^2 + q_4^2}. \quad (\text{B.23})$$

- Complex conjugate:

$$\mathbf{q}^* = \begin{bmatrix} -\bar{\mathbf{q}}_{1:3} \\ q_4 \end{bmatrix}. \quad (\text{B.24})$$

- Identity:

$$\mathbf{q}^{-1} = \frac{\mathbf{q}^*}{\|\mathbf{q}\|^2}. \quad (\text{B.25})$$

Another possible way to represent attitude is obtained simply dividing the vector part of Euler symmetric parameters by its scalar part, as stated in Equation B.26, resulting in a often called *Gibbs vector*, \mathbf{g} , or Rodrigues parameters. It gives the attitude representation on

Equation B.27.

$$\mathbf{g} = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \end{bmatrix} = \frac{\boldsymbol{\eta}}{\eta_4} = \frac{1}{\eta_4} \begin{bmatrix} \eta_1 \\ \eta_2 \\ \eta_3 \end{bmatrix}. \quad (\text{B.26})$$

$$A(\mathbf{g}) = I_3 + 2 \frac{[\mathbf{g} \times]^2 - [\mathbf{g} \times]}{1 + \|\mathbf{g}\|^2}. \quad (\text{B.27})$$

However, the Gibbs vector is not recommended as a global attitude representation, because it is not clearly defined for odd multiples of 180° . A modification of Rodrigues parameters allows this singular point to be moved to odd multiples of 360° , resulting on modified Rodrigues parameters, \mathbf{p} , of Equation B.28 and the attitude representation of Equation B.29:

$$\mathbf{p} = \frac{\boldsymbol{\eta}_{1:3}}{1 + \eta_4}. \quad (\text{B.28})$$

$$A(\mathbf{p}) = I_3 + \frac{8[\mathbf{p} \times]^2 - 4(1 - \|\mathbf{p}\|^2)[\mathbf{p} \times]}{(1 + \|\mathbf{p}\|^2)^2} \quad (\text{B.29})$$