UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS UNIVERSITÁRIO, CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Victor Hugo Schulz

A Verification Platform for Improving the Design and Test of Star Trackers

Florianópolis
2020

Victor Hugo Schulz

A Verification Platform for Improving the Design and Test of Star Trackers

Tese submetida ao Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Santa Catarina para a obtenção do título de doutor em Engenharia Elétrica.
Supervisor:: Prof. Eduardo Augusto Bezerra, Dr.
Co-supervisor:: Prof. Eduardo Todt, Dr.

Florianópolis
2020

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Schulz, Victor Hugo
A Verification Platform for Improving the Design and Test of Star Trackers / Victor Hugo Schulz ; orientador, Eduardo Augusto Bezerra, coorientador, Eduardo Todt, 2020.
86 p.

Tese (doutorado) - Universidade Federal de Santa Catarina, Centro Tecnológico, Programa de Pós-Graduação em Engenharia Elétrica, Florianópolis, 2020.

Inclui referências.

1. Engenharia Elétrica. 2. Sistemas Embarcados. 3. Determinação de Atitude. 4. Verificação. 5. Pequenos Satélites. I. Bezerra, Eduardo Augusto. II. Todt, Eduardo. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia Elétrica. IV. Título.

Victor Hugo Schulz

A Verification Platform for Improving the Design and Test of Star Trackers

O presente trabalho em nível de doutorado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de doutor em Engenharia Elétrica.

Documento assinado digitalmente
Telles Brunelli Lazzarin
Data: 20/07/2020 09:37:22-0300
CPF: 023.400.719-25

Prof. Telles Brunelli Lazzarin, Dr.
Coordenador do Programa

Documento assinado digitalmente
Eduardo Augusto Bezerra
Data: 19/07/2020 18:36:43-0300
CPF: 830.851.577-00

Prof. Eduardo Augusto Bezerra, Dr.
Supervisor:

Florianópolis, 11 de março de 2020.

Este trabalho é dedicado a meu pai, Jorge Gruhn Schulz, que sempre buscou o aperfeçoamento pessoal e profissional no estudo e pesquisa acadêmicos, e que serviu de modelo de inspiração e persistência para que eu pudesse concluir meus estudos com o objetivo de alcançar, como ele, o grau de Doutor.

ACKNOWLEDGEMENTS

I would like to express my gratitude for my supervisor, Prof. Eduardo Bezerra, for the support in this work, helping me from the beginning to the very end of this journey. To my co-supervisor, Prof. Eduardo Todt, for the contribution on defining the subject of the work, which is a continuation of our work on my Master's dissertation. To Prof. Mengu Cho, for receiving me and being my co-supervisor during the portion of this work that was done in Kyutech, Japan. For Professors Sangkyun Kim, Fabian Luis Vargas and Laio Oriel Seman for the excellent work and contributions, which extended beyond the roles of examiners. For the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES) and Ministério da Educação (MEC) for the financial sponsoring that made the realisation of this work possible. For my colleagues in Brazil, Gabriel Mariano Marcelino and Jeferson Santos Barros, who contributed substantially to the contents of this work; and similarly, to Necmi Cihan Örger, Ibukun Oluwatobi Adebolu, and Hirokazu Masui in Kyutech, who aided and collaborated in the realization of the space environment experiments. For the contributions to the quality of the text, I would like to thank Kent Rush and Aleksander Michal Sobieski for the text revisions; and Manfred Kratzenberg, Sara Martinez Vega and Juan José Rojas Hernández for the insightful discussions. I would like to also thank John L. Polansky, George Maeda, Sayo Tsukinari and Atsushi Okubi for the help with the logistics when relocating to Japan, and specially for Juan José Rojas Hernández (once more), Yiğit Çay and Aekjira Kuyyakanont, who received me as good friends and helped me to feel at home.

RESUMO

Não é uma tarefa trivial o processo de desenvolvimento de um sensor de estrelas, desde a fase conceitual até a sua implementação final, passando simultaneamente pela etapa de certificação do sistema. O desafio é ainda maior quando, por razões de flexibilidade, sejam escolhidos para implementação sistemas de processamento do tipo Sistema-em-um-Chip (SoC – *System-on-Chip*) combinando componentes de *software* e *hardware* configurável. Este trabalho propõe a utilização de imagens de estrela sintéticas (um céu simulado), unido à estrutura padronizada da Metodologia de Verificação Universal (UVM – *Universal Verification Methodology*) como base de uma abordagem de desenvolvimento. O objetivo é organizar e acelerar o projeto, melhorar a qualidade do sistema quanto à produção de resultados corretos e oferecer métricas para a comparação de diferentes algoritmos utilizados em sensores de estrelas. O retrabalho potencial futuro é reduzido através de duas formas: nesta tese foi desenvolvido um simulador e plataforma de desenvolvimento que são distribuídos sob licença de software livre; e a estrutura da UVM estimula a reutilização de código através da adoção de uma abordagem orientada a objetos. Está sendo proposta uma estrutura do tipo caixa preta para a plataforma de verificação com interfaces padronizadas, e exemplos foram apresentados sobre como essa abordagem pode ser aplicada ao desenvolvimento de um sensor de estrelas para pequenos satélites, tendo como alvo o desenvolvimento em SoC. As mesmas bancadas de testes (*test benches*) foram aplicadas a ambas as implementações conceituais iniciais (em *software* apenas) como a posteriores implementações de sistemas híbridos (em *software* e *hardware*), em uma configuração Hardware no Laço (HIL – *Hardware-In-the-Loop*). Essa estratégia de reaproveitamento de bancadas de testes também se mostrou interessante ao revelar a capacidade de regressão nos testes realizados através da plataforma desenvolvida. Ainda, o simulador foi utilizado para injetar ruído específico, para que o sistema pudesse ser avaliado em algumas condições ambientais de mundo real.

Palavras-chave: star tracker; sensor de estrelas; verificação; FPGA; field-programmable-gate-arrays; cubesat; pequenos satélites; determinação de atitude.

RESUMO EXPANDIDO

Introdução

Atualmente pode-se observar uma tendência no crescimento do lançamento de nano-satélites (satélites com massa menor que 10 kg). Estes satélites vêm progressivamente substituindo as funcionalidades antes restritas a satélites maiores, o que gera demanda para subsistemas miniaturizados que possuem restrições na energia disponível. Isto ocorre devido à menor área dos painéis solares utilizados em nano-satélites. Um exemplo de tal subsistema, e foco deste trabalho, são os sensores de estrelas (em inglês *star sensors* ou *star trackers*). Sensores de estrelas são responsáveis por determinar a atitude de veículos espaciais ou satélites, ou seja, sua orientação no espaço com relação a um sistema de referência inercial. Os sensores de estrelas se destacam, quando comparados aos demais sensores de atitude, pela maior precisão ($< 0,1°$), funcionamento mesmo em condições de eclipse e por diretamente prover a atitude. As aplicações principais são aquelas que requerem maior precisão: capturar imagens com ângulo de visão restrito e apontar antenas direcionais. Os sensores de estrelas no estado da arte funcionam encontrando a correspondência entre as estrelas visíveis e aquelas presentes em um catálogo interno (identificação). O *hardware* é muito similar ao das câmeras digitais. A principal diferença são os algoritmos de processamento de imagens, e a eficiência desses algoritmos impacta no consumo de energia do subsistema. A redução no número de instruções necessárias para a realização das tarefas reduz a energia necessária. Em alguns casos, otimizações não apenas de software, mas também em hardware podem ser encontradas na literatura. A presença de sistemas com processamento misto traz desafios nas etapas de verificação. A utilização de ferramentas de verificação recentes, tais como a Metodologia de Verificação Universal (UVM) aplicada à linguagem de descrição de *hardware SystemC* alivia estes desafios. Existem trabalhos publicados em que se aplicam estas ferramentas a sistemas de processamento de imagens, o que suporta a aplicação também para sensores de estrelas. Neste trabalho foi criada uma plataforma de verificação para sensores de estrelas utilizando essas ferramentas. A biblioteca *OpenCV* foi também utilizada, juntamente a modelos matemáticos existentes, para a geração de imagens de céu estrelado sintéticas com a introdução de ruído controlado. A ferramenta foi criada para que universalmente possam ser testados diversos tipos (e partes) de algoritmos de sensores de estrelas, tanto isolados como agindo de forma integrada. A ferramenta busca resolver problemas encontrados na avaliação e comparação entre diversos algoritmos, tais como a inexistência de uma ferramenta disponível como software livre, e a falta de padronização na aplicação de testes. A plataforma criada está disponível como software livre, pode replicar configurações utilizadas por diversos autores, permite co-simulação de software e hardware, e trabalha na verificação de sistemas em configuração de caixa preta, facilitando a troca dos dispositivos ou algoritmos que são testados por ela.

Objetivos

O objetivo principal é propor um design para sensores de estrelas miniaturizados com o foco na redução dos requisitos de *hardware* do sistema de processamento. Isto é realizado através de otimizações que refletem na redução do número de instruções necessárias para a execução das rotinas, e também considerando questões do ambiente espacial onde os sensores de estrelas operam. Como objetivo secundário, uma plataforma de verificação foi construída para acelerar o desenvolvimento e medir as melhorias efetivas. Essa plataforma também permite a modelagem de ruídos ambiantais presentes em operações normais de satélites, ou resultantes do lançamento. A plataforma de verificação foi criada de forma universal para que possa servir para a verificação de múltiplos algoritmos utilizados em sensores de estrelas.

Metodologia
Foram implementados de forma completa algoritmos de referência de sensores de estrelas (1. *crescimento de região* como determinação de centroides; 2. *grid* como identificação de estrelas; 3. *QUEST* para determinação final de atitude). O foco principal foi na otimização do algoritmo de *grid* para identificação de estrelas. Este algoritmo pode ter seu banco de dados de características implementado de forma binária, o que permitiu a aplicação de certas otimizações tentando melhorar a sua velocidade e desempenho em termos de estrelas corretamente identificadas. Uma plataforma de verificação foi construída, permitindo medir o impacto destas otimizações, bem como auxiliar na aceleração do desenvolvimento dos algoritmos em si e introduzir diversos tipos de ruídos controlados, o que é necessário para a avaliação do comportamento dos algoritmos em ambientes reais. O algoritmo de determinação de centroides foi também otimizado através de implementação parcial em FPGA (*hardware*), e as melhorias foram medidas pela plataforma.

Resultados e Discussão
Como aplicação da plataforma de verificação no desenvolvimento de sensores de estrelas miniaturizados, foi demonstrada a possibilidade de replicar resultados de outros autores. Ainda, o desenvolvimento do sensor de estrelas foi acelerado através da otimização pontual apenas de partes que possuem alta demanda de processamento, o que foi medido pela plataforma. As otimizações realizadas a partir de implementação híbrida em *software* e *hardware* em FPGA levaram a um ganho de até 123 vezes no desempenho do algoritmo de centroide empregado. No caso do algoritmo de identificação de estrelas, ganhos de até 12.5 vezes na velocidade de processamento foram atingidos através da aplicação de otimizações puramente em software. A maneira como o algoritmo de *grid* realiza a busca no banco de dados interno de padrões de estrelas foi otimizada. Testes que replicam o ambiente espacial foram realizados em sensores óticos comerciais que são candidatos a utilização com sensores de estrelas. Estes foram testes de vibração, choque mecânico e dose total ionizante. As variações observadas nos sensores reais puderam ser reproduzidas na forma de ruído nas simulações realizadas pela plataforma de verificação, o que permitiu a avaliação do comportamento dos algoritmos em condições de mundo real. Modificações na função que pontua a semelhança entre padrões do algoritmo de *grid* levaram também a um aumento em $1,5\%$ na taxa de determinação correta de atitude, porém com um impacto negativo de 50% no tempo de execução. Finalmente, foi possível automatizar uma série de testes de avaliação dos algoritmos para que possam ser executados em sequência, acelerando a obtenção de dados.

Considerações Finais
A plataforma de verificação apresentada pela presente tese trouxe diversas contribuições quando comparada a ferramentas existentes. Exemplos são a utilização da estrutura padronizada da UVM, a utilização de uma estrutura modular incentivando a reutilização de componentes de verificação, a possibilidade de verificar subcomponentes de *software* de sensores de estrelas forma isolada ou conjunta, a aceleração e auxílio no desenvolvimento e otimizações, e a habilidade de reproduzir diversas condições utilizadas em testes por outros autores. O compartilhamento do simulador e plataforma de verificação implementados como *software* livre também é uma contribuição, o que permite sua reutilização e melhora pela comunidade científica. Isso permite que futuramente sejam realizados trabalhos visando a padronização dos procedimentos de testes, já que o ambiente de testes pode ser mantido o mesmo utilizando a plataforma compartilhada.

Palavras-chave: star tracker; sensor de estrelas; verificação; FPGA; field-programmable-gate-arrays; cubesat; pequenos satélites; determinação de atitude.

ABSTRACT

Proceeding from the conceptual phases of the development of a star tracker, until a complete working system is produced, while simultaneously ensuring the correctness of the approach, is not a trivial task. The challenge can be increased when, for flexibility reasons, the processing system is implemented through a System-on-Chip (SoC) combining pieces of software and configurable hardware. This work proposes the use of synthetic star images (a simulated sky), allied with the standardised structure of the Universal Verification Methodology (UVM) as the base of a design approach. The aim is to organise the project, speed up the development time, improve the correctness of the system, and provides metrics for the comparison of different algorithms. Future rework is reduced through two methods: we developed a simulator and verification platform that are shared under a free software licence; and the layout of UVM enforces reusability of code through an object-oriented approach. We propose a black-box structure for the verification platform with standard interfaces, and provide examples showing how this approach can be applied to the development of a star tracker for small satellites, targeting a SoC design. The same test benches were applied to both early conceptual software-only implementations, and later optimised software-hardware hybrid systems, in a hardware-in-the-loop configuration. This test bench reuse strategy was interesting also to show the regression test capability of the developed platform. Furthermore, the simulator was used to inject specific noise, in order to evaluate the system under some real-world conditions.

Keywords: star tracker; star sensor; verification; star simulator; FPGA; field-programmable-gate-arrays; cubesat; small satellites; attitude determination.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AMS	Analog Mixed-Signal
ARM	Advanced RISC Machine
ASCII	American Standard Code for Information Interchange
ASIC	Application-Specific Integrated Circuit
CCD	Charge-Coupled Device
CMOS	Complementary Metal-Oxide Semiconductor
COTS	Commercial Off-The-Shelf
CPLD	Complex Programmable Logic Device
CSI-2	Camera Serial Interface 2, specification from MIPI
DUT	Device Under Test
EKF	Extended Kalman Filter
ESA	European Space Agency
ESL	Electronic System Level Design
FIR	Finite Impulse Response (filter)
FOV	Field of View
FPGA	Field-Programmable Gate Array
GPS	Global positioning system
HDL	Hardware Description Language
ICRS	International Celestial Reference System
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organisation for Standardisation
ISS	Instruction Set Simulator
JPL	Jet Propulsion Laboratory (NASA)
LEO	Low Earth Orbit
LSB	Least significant bit
LUT	Lookup Table
M-12	ISO metric screw thread of 12mm of diameter
MIPI	Mobile Industry Processor Interface
MSB	Most significant bit
NASA	National Aeronautics and Space Administration
NEON	Advanced SIMD Instructions
PCB	Printed Circuit Board
PSF	Point Spread Function
QUEST	Quaternion Estimator
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level
SIMD	Single Instruction, Multiple Data

SNR	Signal to Noise Ratio
SSE	Streaming SIMD Extension
TCP/IP	Transmission Control Protocol/Internet Protocol
TID	Total Ionizing Dose
TLM	Transaction Level Modeling
UVC	Universal Verification Component
UVM	Universal Verification Methodology
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

LIST OF SYMBOLS

DE_{rad}	Declination in Radians
DN	Brightness
HIP	Hipparcos Identifier
M_{max}	Maximum visual magnitude of stars
$POPCNT$	Population count instruction
RA_{rad}	Right Ascension in Radians
ROI	Region of interest
R	Reference stars
$VCNT$	Vector Count set bits
V_{mag}	Visual Magnitude
V	Visible stars
XOR	Exclusive OR operator
α	Right ascension
$\&$	Logical AND operator
c_x	Centre of image sensor plane (x)
c_y	Centre of image sensor plane (y)
δ	Declination
$erf(x)$	Error function
e	Acceptable measurement error
f_x	Focal length (x coordinate)
f_y	Focal length (y coordinate)
f	Focal length
γ	Closest neighbor angle
g	Grid size
μ_α	Proper motion of right ascension
μ_δ	Proper motion of declination
m	Binary mask
$NXOR$	Negated Exclusive OR operator
pat	Pattern
$pmDE$	Proper Motion in Declination
$pmRA$	Proper Motion in Right Ascension
pr	Pattern radius
p	Quaternion representation of unit vector
q	Rotation Quaternion
res	Resolution of image sensor
r	Reference star
$\tan()$	Tangent trigonometric operator
θ	Angle of rotation

u	Unit Vector
x_c	Centroid coordinate (x)
y_c	Centroid coordinate (y)

CONTENTS

1 INTRODUCTION

The CubeSat, initially proposed by the California Polytechnic State University, became a standard format for nanosatellites and picosatellites. CubeSats are composed of units of 10x10x10 cm cubes (1U), with each unit weighing up to 1.33 kg. They were firstly developed with the academic goal of facilitating the design and test of spacecraft technologies by students. By following the specifications, it is possible to standardise the orbital deployers (MEHRPARVAR et al., 2014), what in practice results in an overall lessening in deployments costs, with an ever increasing participation of commercial and amateur projects in addition to the academic designed CubeSats. A classification of satellites based on their mass is shown in Table 1.

Table 1 – Classification of satellites by mass.

Type	Mass(kg)
Conventional large satellite	> 1000
Medium satellite	500-1000
Mini-satellite	100-500
Micro-satellite	10-100
Nano-satellite	1-10
Pico-satellite	< 1
Femto-satellite	< 0.1

Source: Adapted from Gao, Clark, et al. (2009).

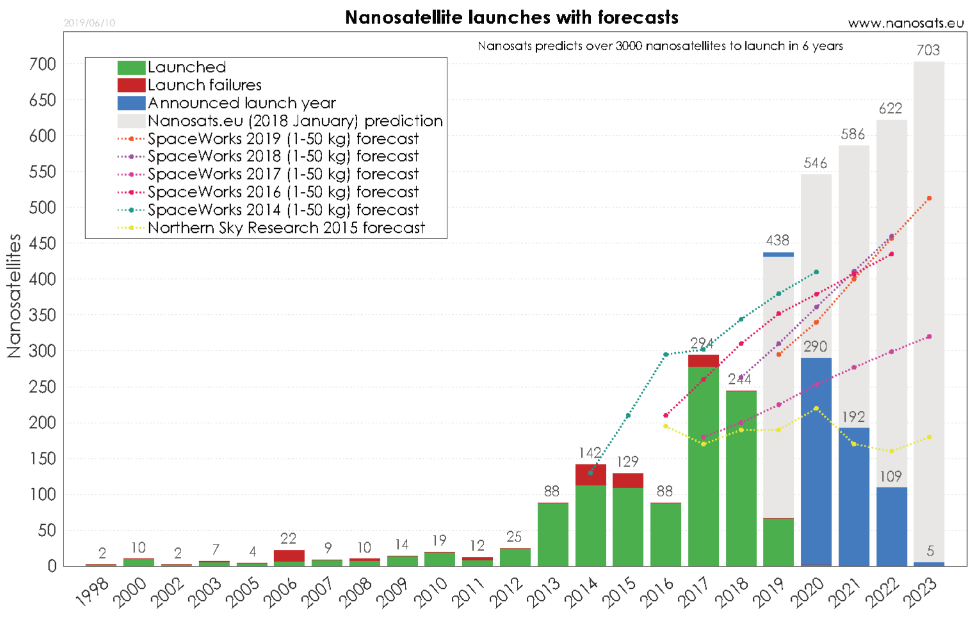
The number of nanosatellites and picosatellites being launched have increased over the years, as shown in Figure 1. Associated with this growth, these smaller satellites have incrementally being able to replace functions previously only performed by bigger satellites, by the means of miniaturisation of their components. The reduced physical volume also implies in smaller solar panels and batteries being employed, resulting in stricter energy constraints for the satellite subsystems. These constraints have brought a demand for technology development toward optimising size, mass and energy consumption of CubeSat components.

As an example which is the focus of this work, a subsystem might be responsible for attitude determination, which is finding the satellite's orientation in space with respect to a given reference system. Reliable attitude information is required, among other uses, for pointing the satellite's solar panels towards the Sun [1] and its antennas to the Earth [2].

There are several types of attitude sensors, which can be used alone or associated for obtaining complimentary measurements and redundancy. Examples of commonly used ones are

[1] In order to optimally generate energy.
[2] Directional antennas can be used to improve gain and reduce the required transmission power.

Figure 1 – Number of Nanosatellites by announced launch years up to June, 2019. Here, the term Nanosatellites also implies the counting of Picosatellites and all CubeSats.



Source: Reproduced from Kulu (2019).

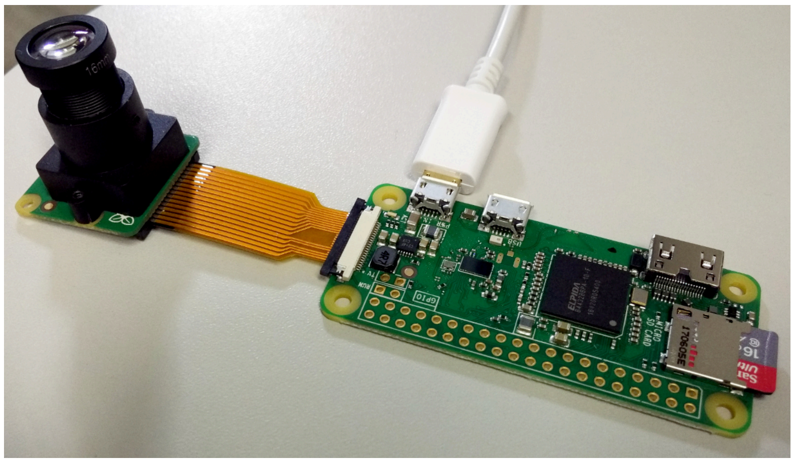
magnetometers, Sun sensors, horizon sensors and star trackers[3]. In comparison, star trackers have up to two orders of magnitude better accuracy in measurement than the other sensors, can work throughout the orbit, including in eclipse conditions, and directly provide an attitude estimation, without relying on extra processing for computing it from measurement vectors (ERLANK, 2013). Satellite missions that employ high resolution optical instruments with long focus focal lengths or those missions that use high gain directional antennas are examples of applications of satellites where a star tracker is often a requirement in order to achieve high attitude accuracy (better than 0.1 degrees) (ERLANK; STEYN, 2014).

State of the art Star Trackers, in general, work by matching (or identifying) the tracked stars with entries in an internal database. This information is then used to determine the attitude, usually by the means of a static attitude determination algorithm[4]. They are physically composed in hardware by a CCD (charge-coupled device) or CMOS (complementary metal-oxide semiconductor) digital image sensor and a lens, a lot similar to what composes a digital camera (MARKLEY; CRASSIDIS, 2014). A prototype of a miniaturised star tracker can be seen in Figure 2.

In star trackers, some power requirements are related to the sensor itself, which depends on the technology used for its construction, while the remaining energy consumption is associated with the image processing system, generally in the form of microprocessors and/or

[3] Star trackers are also sometimes called star sensors.
[4] Static attitude determination works without memory, while dynamic attitude determination would depend on previous states.

Figure 2 – Prototype of a Star Tracker produced on GSE/UFSC.



Source: Elaborated by the author.

programmable logic (for example FPGAs or CPLDs).

Algorithm optimisations in software and hardware form can reduce the requirements in terms of speed and energy by reducing the necessary processing clock cycles in order to perform comparable computations. These enhancements are specially desirable for star trackers embedded in smaller satellites.

After an image is acquired by the CCD or CMOS sensor, there are dedicated algorithms which are responsible for the image processing, identification of the stars present in the image and the attitude determination from this data. Thus, star tracker processing computations can be separated in three parts: subpixel centroiding; star identification; and static attitude determination, which are individually described in chapter 3.

During the development of star tracker algorithms, it is necessary to evaluate how well the system behaves. Significant input needs to be generated, and the output response needs to be checked. While traditionally the algorithms implementation was done in software, there are recent examples where hardware implementations targeting programmable logic were explored (ZHOU et al., 2016; ZHAO et al., 2017), bringing advantages in terms of throughput.

Thus, the verification of star trackers has to consider the possibility of the system being composed of software, hardware, or a combination of both. This ability is also important in this context of top-down design, when the engineer starts with a conceptual high-level software implementation that is progressively specialised and optimised for the target hardware and application.

Modern verification tools can be employed to address these emerging challenges. The Universal Verification Methodology (UVM), IEEE 1800.2 standard, applied to the SystemC family of libraries[5], offers a standardised structure for constructing an environment that supports

[5] `https://accellera.org/activities/working-groups/systemc-verification/uvm-systemc-faq`

the co-verification of software and hardware. The software part is usually written in the C++ language, and the hardware also described directly in C++, with the help of SystemC. More traditional hardware description languages (HDLs) such as VHDL and Verilog can also be used in co-verification. Once a test bench is created, it can be reused without changes during top-down design. Therefore, the algorithms under development can be directly compared, as they evolve. As opposed to traditional HDLs, multiple C++ libraries can be used to simplify the high-level development of the test benches or the system itself, saving time. UVM-SystemC is a recent development in the field of verification. It is currently a draft under public review, with its first public version being released in 2016 by Accellera.

There are works where computer vision applications successfully employed UVM-SystemC as a design aid tool. In Mefenza, Yonga, and Bobda (2014), a verification environment using SystemC and UVM was created for computational demanding video-based embedded systems. A system design starting with an executable specification in C++ and the computer vision library OpenCV[6] was verified. The system was progressively refined into lower levels of abstraction as an FPGA based smart camera. The final system works in a Zynq-7000 SoC, with the software part running in an ARM processor and the hardware part in the device's programmable logic. Similarly, in Campos et al. (2017) a UVM-SystemC environment was used to build a framework for the design and validation of face detection systems. Differently, no hybrid system is considered, but instead, a high-level model developed using OpenCV is used as the golden reference model, from which a complete hardware implementation of the system in SystemVerilog was compared.

In the present work, we also used the UVM-SystemC environment paired with the OpenCV computer vision library as the base of our verification platform. The main distinction is that our approach targets a different area of research: the development of star trackers. Overall, the similarities in the tools employed support the idea that the methodology can successfully be applied for the design of star trackers, which can be understood as computer vision systems.

As the nature of the input images of our system is different, we developed a star simulator that is able to synthesise the required stimulus: synthetic sky images with added controllable noise. A mathematical model for this type of image synthesis is presented in Hua-Ming, Hao, and Hai-Yong (2015) and Guangjun Zhang (2017). The general skeleton of the process we follow in this work is very similar. The main difference is in the way the rotations are performed: we used quaternions instead of Euler's rotation matrices. The use of quaternions simplifies the equations and avoids gimbal lock limitations.

Commonly, sky simulators allow the introduction of controllable noise to the system. For example, in the previously mentioned models, Gaussian noise was used to model random background noise. A more thorough model of the optical system of a star tracker, which includes the lens and image sensor, can be seen in Knutson (2012). The simulator considers the physical aspects of the optical system for emulating noise. In our verification platform we employed

[6] `https://opencv.org/`

a practical approach which uses simplified noise models (not much different than the former solutions). The mathematical functions for noise generation have their parameters tweaked in order to behave similarly to a physical optical system used as a reference. In the physical system, the noise was measured in conditions similar to what is expected during operation.

In the usual way star tracker algorithms are evaluated, the images produced by a star simulator are employed as the input of one or more reference algorithms. The same input is then applied to the novel algorithm, and the results compared with the intent of showcasing improvements. Kolomenkin et al. (2008), when discussing this matter, stated that this is not a trivial task, due to no agreed standard: "Many authors have referred to different aspects of star tracker performance such as speed, accuracy, memory requirements, and stability. But each of them used a different configuration". The consequence of their observation can be seen in Figure 3. It shows three different authors' (PADGETT; KREUTZ-DELGADO; UDOMKESMALEE, 1997; ZHANG; WEI; JIANG, 2008; NA; ZHENG; JIA, 2009) data on how the Grid algorithm (PADGETT; KREUTZ-DELGADO, 1997) behaves with the presence of positional noise, using different test configurations for field of view (FOV), resolution (*res*), maximum visual magnitude (M_{max}), and grid size g. Different star simulators were also used, and it is likely that the implementation of Grid differs between researches.

Instead of suggesting a given set of configurations that should be followed by researchers when evaluating star trackers, our contribution is to provide a universal verification platform and star simulator. It has the flexibility of easily working with any desired conditions. Thus, our verification platform was built with configurable optical parameters such as sensor resolution, field of view and maximum visible star magnitude. Noise levels are also configurable. The aim is to easily test algorithms in similar conditions of those used in multiple, different previous researches. Effectively, this reduces the need of implementing again reference algorithms.

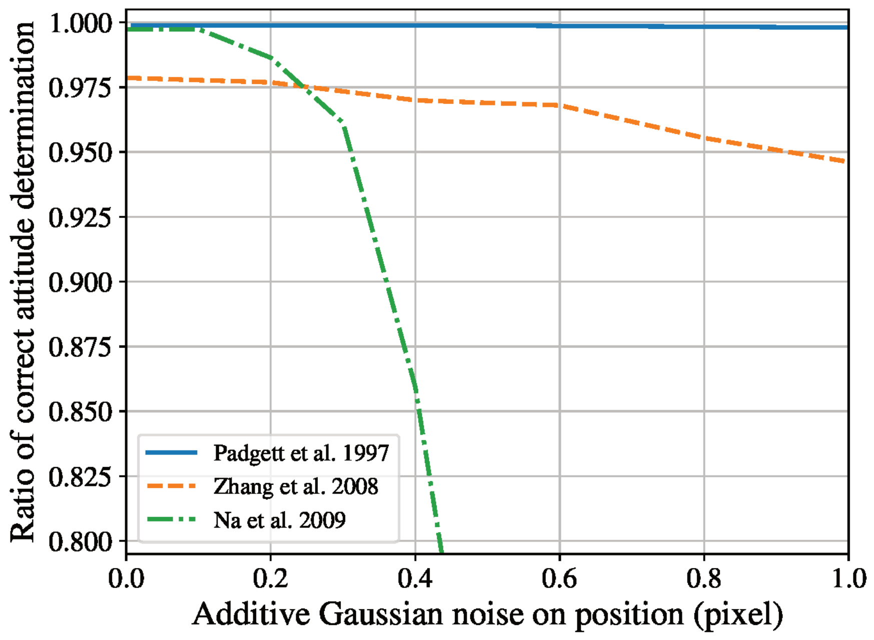
If widely adopted, our universal verification platform could become a common tool, improving the reproducibility of results. Therefore, all the code of the simulator and verification platform is being shared[7] under the Apache Licence Version 2.0. Thus, the platform's structure, the star simulator and test cases can be reused to evaluate different algorithms. To simplify interfacing, we proposed a black-box structure following the existing UVM standard, allowing easier collaboration. Universal interfaces were created so that it becomes possible to test the centroiding, star identification, and attitude determination algorithms separately or working together.

1.1 GENERAL OBJECTIVE

This work has the main objective of proposing a design for miniaturised star trackers, with the focus on reducing the hardware requirements of the processing system. This reflects in smaller energy requirements through the appropriated sizing of the latter for a computational

[7] `https://github.com/schulz89/Verification-Platform-for-Star-Trackers`

Figure 3 – Effect of different test configurations on the behaviour of the Grid Algorithm. Padgett, Kreutz-Delgado, and Udomkesmalee (1997) used $FOV = 8° \times 8°$, $res = 512 \times 512$, $M_{max} = 7.5$, $g = 40$; Zhang, Wei, and Jiang (2008) used $FOV = 12° \times 12°$, $res = 1024 \times 1024$, $M_{max} = 6.0$, $g = n/a$; Na, Zheng, and Jia (2009) used $FOV = 8° \times 8°$, $res = n/a$, $M_{max} = 6.5$ and $g = 40$.



Source: Data from Padgett, Kreutz-Delgado, and Udomkesmalee (1997), Zhang, Wei, and Jiang (2008), and Na, Zheng, and Jia (2009) was replotted by the author.

simpler task, which is of relevant importance considering its application in small satellites (i.e. through reducing the number of required software instructions, the microprocessors employed can be simpler; or can work with reduced frequency when idle). For this specific application, the development took in consideration the environmental conditions of LEO[8], with specific routines for tolerance to failure and noises being considered due to the existing adverse effects during operation.

In order to speed up the development, the system behavioural verification and the proposed improvements in software that aim for increased energy efficiency, as a secondary objective, a standardised verification system responsible for the synthesis of the inputs of the star tracker device was built. The environmental conditions of normal satellite operation, such as the introduction of specific noise patterns due to the exposition to solar radiation and variation in optical parameters of the lens/sensor set due to vibration and shock during launch were modelled and introduced into the simulation procedures performed by the verification system.

[8] Low Earth Orbit.

1.2 SPECIFIC OBJECTIVES

Considering the general objectives, the following specific objectives are defined:

1. Explore different star identification algorithms as candidates for use in miniaturised star trackers than those already chosen by other works;

2. Propose improvements in the selected algorithm, focusing on the downsides pointed out by the scientific community;

3. Evaluate the proposed improvements with tests and metrics that are commonly adopted with similar algorithms for star identification;

4. Study of standardised testing platforms employed within different research areas and contexts for serving as a structural blueprint for the proposed testing platform;

5. Generalise the test platform so that it can be used to qualify other algorithms, serving as a proposed standardised universal testing platform;

6. Publish the testing platform specifications and the companion software in the form of free software.

1.3 RESEARCH QUESTIONS

This Section presents the research questions that serve as direction for this work's development, supporting the objectives.

1. Could the speed advantages of working directly with binary descriptors be explored in star identification algorithms of the pattern recognition class[9], making them better candidates for use in miniaturised star trackers? In the related subject of *Local Features*, which is also a subclass of pattern recognition problems, the matching of binary descriptors was shown to be performed faster on modern microprocessors when compared to the more established vector-based descriptors, due to the available processor instructions for Boolean operations[10] (MUJA; LOWE, 2012). Could a similar performance gain be achieved within the context of star identification?

2. By doing the pattern matching in the binary format, instead of the usual lookup table[11] based approach, could the Boolean matching function be optimised to better use the available information from the catalogue and sensor descriptors? Existing algorithms of the pattern recognition class did not yet explore this possibility (ZHANG; WEI; JIANG, 2008; ZHAO et al., 2017).

[9] The classes for star identification algorithms are described in subsection 3.2.1
[10] subsection 3.2.2 discusses the parallel with *Local Features* and the specialised instructions.
[11] The binary and lookup table formats are described in subsection 3.2.4.

3. Could some techniques used to optimise algorithms from the subgraph isomorphism class be reapplied to algorithms of the pattern recognition class? Would this result in reduction in complexity and resource requirements? An investigation of which of those techniques would benefit the optimisations for miniaturisation is necessary.

4. When constructing a verification platform, could the inputs and outputs be made universal, so that it would be possible to test software components isolated (e.g. centroiding, star identification or static attitude determination) or associated?

5. How can the design of star trackers be accelerated by the use of a verification platform?

1.4 METHODOLOGY

Based on the research objectives, the complete software stack of a star tracker was implemented in order to be used as a reference. For the star identification part, the *grid algorithm* (PADGETT; KREUTZ-DELGADO, 1997) was selected as the basis in which the implementations are proposed. The algorithm was chosen due to its widely recognition as the reference algorithm from which the various other pattern recognition algorithms were derived, and due to its star patterns being defined in the form of binary descriptors, which are targets for optimisation in computational efficiency during processing.

State of the art algorithms benefit from the acceleration of processing through dedicated hardware implementation using binary descriptors (ZHAO et al., 2017). These algorithms, derived from the *grid algorithm*, can also be improved by the same proposed optimisations, without the need of significant modifications.

An analysis of related publications revealed two main criticism about the grid algorithm. The first is that its database search procedure is asymptotically worse than newer algorithms of the subgraph isomorphism class (SPRATLING; MORTARI, 2009). The second is that it usually requires a large field of view (FOV) for correct matching (NA; JIA, 2006; HO, 2012), as the attitude determination can be compromised when only a few stars are visible. Considering that the same sensor is used, when working with a larger FOV, the number of pixels between two given stars are fewer, impacting negatively the accuracy of measuring their angular distance.

By organising the grid algorithm's star database in a binary form and working with binary descriptors, it was possible to propose a few improvements with the existing criticisms in mind, which can impact positively in the algorithm's speed and expand the sky coverage by improving the number of correct matches between sensor derived patterns and the database entries.

An evaluation platform was built in order to verify the implementations and measure the actual improvements, comparing the algorithm with introduced modifications with the reference implementation. As the embedded system which composes a star tracker is mixed between software and hardware, the verification platform was implemented following the IEEE 1800.2-2017 standard (IEEE, 2017), using the Universal Verification Method (UVM) structure and using the system modelling language SystemC, which allows for the co-verification of mixed systems with software and hardware, including the support for hardware-in-the-loop structures.

The exposition environments where radiation is present, for example during orbit operation of a star tracker, can cause effects over the images captured by the CMOS sensor in the form of dark current, and random telegraph signal (HOPKINSON; MOHAMMADZADEH; HARBOE-SORENSEN, 2004; VIRMONTOIS et al., 2014). In parallel, cyclic thermal deformations can distort the focal plane of the optical system during operation (SAMAAN; MORTARI; JUNKINS, 2006). Such effects were measured in hardware using a prototype, and subsequently mathematically modelled in order to be introduced in the synthetic images produced by the verification system.

This platform was constructed with generalisation in mind, so that other star identification algorithms could also be evaluated by the same benchmarks, thus serving as a contribution for standardising star identification performance analysis. The associated metrics were designed to be able to universally evaluate star identification algorithms, so that they can serve as a common ground of comparison between different implementations.

Linking the verification platform with the star tracker design steps, we considered the possibility of speeding up development through a top-down design. It begins with the high-level algorithms being implemented in a computer. Development is simplified by using C++ libraries, such as OpenCV 3. The system evolves down to lower levels of abstraction, becoming a hybrid system. Parts of the algorithm that are identified to be more demanding in terms of processing power are then optimised and implemented in hardware using FPGA stream processing. As lower level implementations get more specialised, C++ libraries can be dropped in exchange for targeted code. As the system evolves, the same test cases that were developed simultaneously with in higher levels of abstraction are reused to ensure correctness of the design in lower levels (regression test).

1.5 ORIGINAL CONTRIBUTIONS

The original contribution of this work is:

- A verification platform for star tracker algorithms, following the structure of the Universal Verification Methodology standard, was designed and shared as free software.

Other contributions of this work include:

- The platform can be used to test different algorithms for star trackers that perform centroid extraction, star identification and attitude determination, separated or acting together. A universal design was achieved through a black box design and well-defined interfaces;

- Direct comparison with data of different authors can be done, as simulations can easily run in different conditions by adjusting the optical parameters and noise levels of tests;

- The platform can be used as an aid to speed up development. A top-down design of mixed software and hardware components is explored as a proof of concept;
- Specific noise simulations can also be performed. A prototype was submitted to satellite launch environment conditions, and the measured noise was used to configure the simulator;
- An initial batch of standard tests that enable direct comparison between algorithms of different authors was created. This is an initial step into creating a standard for comparing algorithms;
- An evaluation of the performance effects of working directly with binary descriptors for pattern based star identification;
- An evaluation of the impact of the proposed modifications for database search complexity reduction and modified scoring function for star identification algorithms working with binary descriptors.

1.6 OUTLINE

The work is organized as follow: section 2.1 presents a brief description of coordinate systems used for attitude determination. This chapter can be skipped if the reader is familiar with the topic. In sequence, section 2.2 describes how miniaturized star trackers work from the hardware perspective, with commercial and academic examples. After, chapter 3 discusses the software components of a star tracker, with focus on the algorithms relevant for this work. In chapter 4, the verification platform based on UVM/SystemC that was created is presented, describing how the star simulator was implemented, with an analysis of the common tests that are done to evaluate star tracker algorithms. In chapter 5, the practical applications of the developed verification platform for speeding up development and optimising star identification algorithms are discussed. The chapter also demonstrates how the platform can be used along with traditional satellite environment tests to evaluate algorithms in the presence of real world noise. Finally, chapter 6 presents the conclusions of the work, remarks and what could be expanded in future works.

2 BACKGROUND

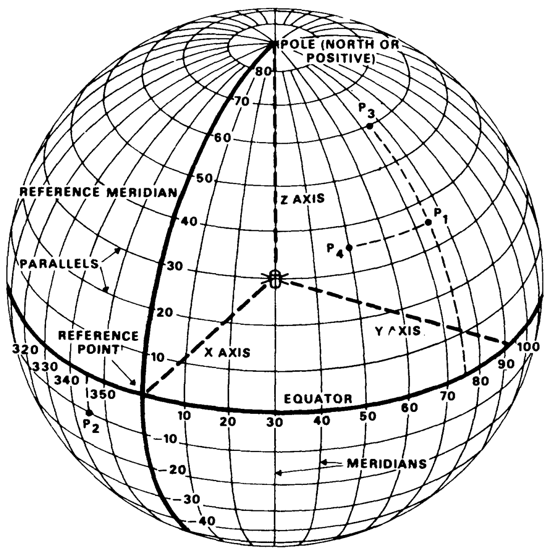
2.1 COORDINATE SYSTEMS FOR ATTITUDE DETERMINATION

Attitude determination is the process of obtaining spacecraft orientation in space. More precisely, the orientation is to be determined relative to an inertial frame of reference or a specific object, such as the Earth or the Sun. The problem of attitude determination can be understood as a geometric problem on a two-dimensional celestial sphere. In the celestial sphere, two angles are necessary to represent a point, and the radius is considered to be unitary (WERTZ, 1978).

In this work, two coordinates systems are used: spacecraft-fixed; and inertial. The former being referenced at the spacecraft itself, and the latter being Earth referenced. They are explained in more details next.

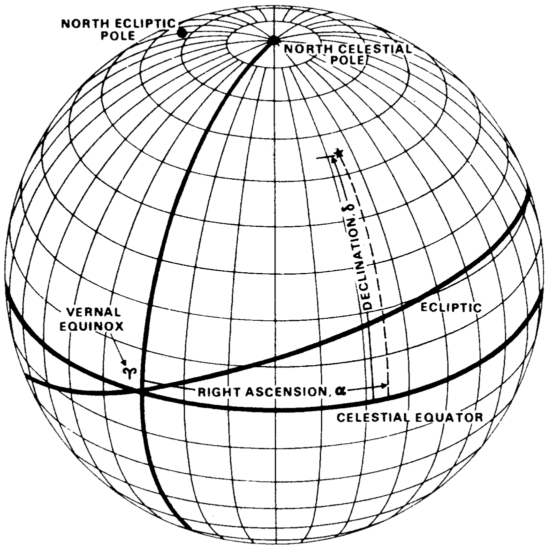
Spacecraft-fixed coordinates is the system in which the measurements are taken, i.e. from the perspective of the spacecraft itself. The two angles that compose the celestial sphere (Figure 4) are called azimuth (the angular distance around the equator between the meridian passing through the relevant point and the reference meridian) and elevation (arc length distance above or below the equator of the celestial sphere). For attitude sensing hardware, the field of view of the sensor is taken as the reference for centring the celestial sphere (WERTZ, 1978).

Figure 4 – Celestial sphere representing the spacecraft-centered coordinate system.



Source: Reproduced from Wertz (1978).

Figure 5 – Celestial sphere representing the inertial (celestial) coordinate system.



Source: Reproduced from Wertz (1978).

The most common inertial coordinate system takes the rotation axis of the Earth as the reference. The North and South poles of the celestial sphere are defined to be parallel with the rotation axis of the Earth, and the reference meridian, called the vernal equinox, is defined as the point where the ecliptic, which is the plane of the Earth's orbit around the Sun, crosses the equator going from south to north. This would be a direction parallel to the line from the centre of the Earth to the Sun on the first day of spring. The idea is that this reference system would be inertial, i.e. fixed relative to the mean position of the stars in the vicinity of the Sun, but in fact changes over time due to gravitational interaction between the Earth, the Moon and the Sun, a phenomenon called precession of the equinoxes. Thus, an inertial coordinate system with the Earth rotation axis must have a date attached in order to accurately define the position of the vernal equinox. The two angles that define the celestial sphere are here called ascension and declination (analogous to the previous azimuth and elevation angles), and they are depicted in Figure 5 (WERTZ, 1978).

Since the celestial sphere has a unit radius, the coordinates can also be mathematically represented as a x, y, z rectangular system, in the form of unit vectors. This form of representation facilitates the expression of mathematical concepts in equations, and is mostly used in this work. Whenever a unit vector is used in this fashion, the reference is specified as either spacecraft-fixed or inertial. The problem of determining the attitude from the sensors then becomes determining the rotation between these two coordinate systems – spacecraft-fixed and inertial, which can be mathematically expressed in various forms: Euler axis + angle; rotation vector; rotation quaternion; Rodrigues parameter (and its modified version); and Euler angles. A comprehensive description of these systems of attitude representation and the conversion between them is available in Markley and Crassidis (2014).

2.2 MINIATURISED STAR TRACKERS

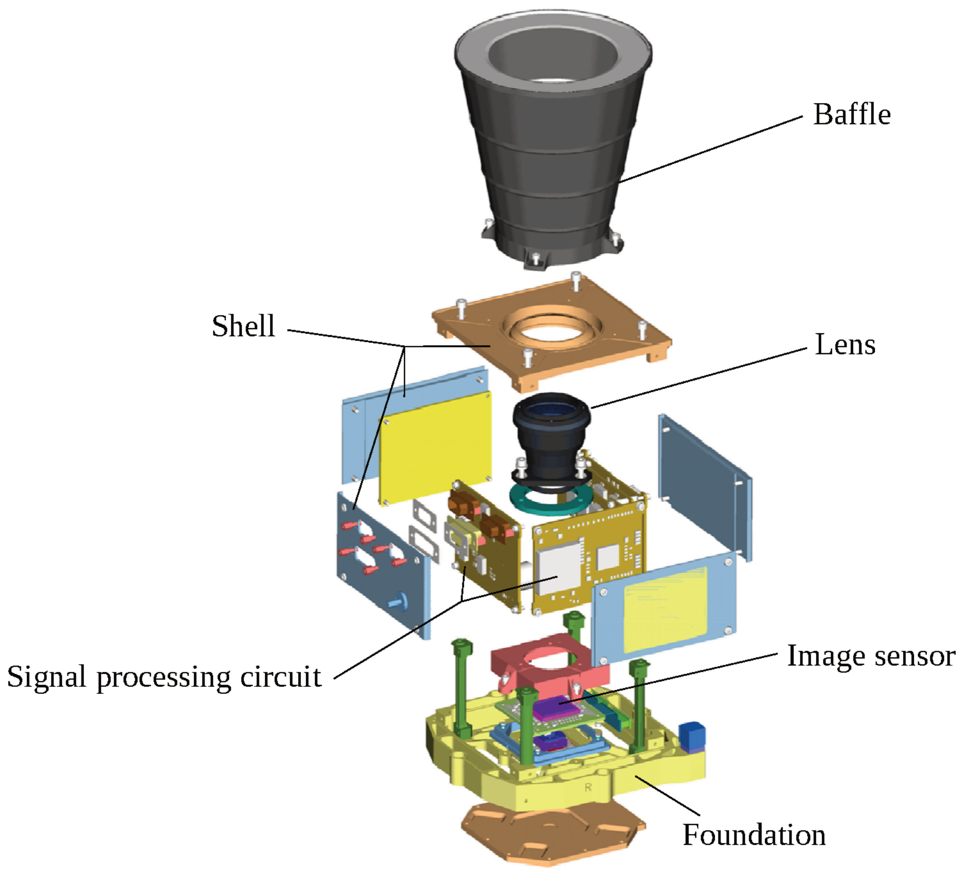
2.2.1 Physical Construction

State of the art star trackers are composed of two main components: a digital image sensor, using the CCD or CMOS technology, and a lens. This configuration is a lot similar to what composes a digital camera (MARKLEY; CRASSIDIS, 2014). An example of an internal structure of a star tracker can be seen in Figure 6. In order to prevent stray light rays from introducing errors in the images, a mechanical construct called baffle is used, which allows only light coming from the front to enter the lenses.

In the same way that smart digital cameras have on-board processing, it makes sense to consider the hardware responsible for processing the image coming from the optical sensor as part of the star tracker. Due to the small quantities in which star trackers are manufactured, usually this processing hardware is present in the form of microprocessors or programmable logic, i.e. FPGAs (Field Programmable Gate Arrays). Manufacturing specific ASICs for this purpose is cost-prohibitive at current production volume.

In the later years, due to the increase in the market of smartphones with embedded cameras, CMOS sensors became available in very small form factors, with increased resolution, sensibility, lower costs and reduced energy consumption. The CMOS sensors also have, in comparison with CCD sensors, better immunity to radiation (MARKLEY; CRASSIDIS, 2014). CCDs were sometimes selected in the past for being able to capture whole images at the same

Figure 6 – Internal structure of ASTRO APS star sensor.



Source: Redrawn from Guangjun Zhang (2017) and Schmidt (2005).

time (global shutter). In most CMOS sensors, every pixel is acquired at a slightly different time (rolling shutter). Nowadays there are commercial CMOS sensors without this limitation.

For miniaturised star trackers, it then makes sense to prefer CMOS sensors due to the stricter energy requirements, size and tolerance in adverse environment conditions. The lenses and mechanical structure (including the baffle) dominate the size and weight of a state of the art star tracker.

2.2.2 Literature Review

There are multiple examples of miniaturised star trackers being produced, which are compatible with size, weight and energy requirements of CubeSats. They are created for both commercialisation purposes and academic research initiatives. Table 2 presents several miniaturised star trackers, including comparison data, which shows that there exist star trackers which have power requirements below 1W, weight less than 0.5 kg and have update rates between 1 and 10 Hz.

Table 2 – Examples of star trackers for nano-satellites and pico-satellites.

Name (Manufacturer)	**Power**	**Accuracy**	**Frequency**
CubeStar (Stellenbosch University)	$0.350W$ (avg.), $0.55W$ (peak)	$< 0.01°$ (c.b.), $< 0.03°$ (a.b.)	$1Hz$
STELLA (Julius Maximilian Univ. of Würzburg)	$0.275W$ (avg.)	$0.01°$ (c.b.), $0.04°$ (a.b.)	$4Hz$
ST-16 (Sinclair Interplanetary)	$< 0.5W$ (avg.), $1.0W$ (peak)	$< 10''$ (c.b.), $< 74''$ (a.b.)	$2Hz$
ST-16RT (Sinclair Interplanetary)	$< 0.5W$ (avg.), $1.0W$ (peak)	$< 4''$ (c.b.), $< 30''$ (a.b.)	$2Hz$
MIST (Space Micro)	$< 3W$ (avg.)	$30''$ (1σ)	$10Hz$
ST-200 (Berlin Space Technologies)	$220mW$ (avg.), $650mW$ (peak)	$30''$ (3σ)	$1Hz$
Nano Star Tracker (Blue Canyon Technologies)	$0.75W$ (avg.), $< 1.0W$ (peak)	$6''$ (1σ, c.b.), $40''$ (1σ, a.b.)	$5Hz$
TS Nano (Blue Canyon Technologies)	$0.75W$ (avg.), $< 1.0W$ (peak)	$6''$ (1σ, c.b.), $40''$ (1σ, a.b.)	$5Hz$

Name (Manufacturer)	**Dimensions**	**Mass**	**Reference**
CubeStar (Stellenbosch University)	$46 \times 33 \times 70mm$ (w. baffle)	$90g$	(ERLANK; STEYN, 2014)
STELLA (Julius Maximilian Univ. of Würzburg)	$91 \times 46 \times 58mm$ (w. baffle)	$167g$	(BARSCHKE et al., 2013)
ST-16 (Sinclair Interplanetary)	$59 \times 56 \times 32mm$ + Ø $48 \times 48.5mm$ (baffle)	$85g + 35g$ (baffle)	(SINCLAIR INTERPLANETARY, 2015a)
ST-16RT (Sinclair Interplanetary)	$62 \times 56 \times 38mm$ + Ø $57 \times 47mm$ (baffle)	$158g + 30g$ (baffle)	(SINCLAIR INTERPLANETARY, 2015b)
MIST (Space Micro)	$< 100 \times 100 \times 50mm$ ($0.5U$; w. baffle)	$500g$ (w. baffle)	(SPACE MICRO, 2015)
ST-200 (Berlin Space Technologies)	$30 \times 30 \times 38.1mm$ (w.o. baffle)	$50g$ (w.o. baffle)	(BERLIN SPACE TECHNOLOGIES, 2012)
Nano Star Tracker (Blue Canyon Technologies)	$100 \times 55 \times 50mm$ (w. baffle)	$350g$	(BLUE CANYON TECHNOLOGIES, 2015b)
TS Nano (Blue Canyon Technologies)	$100 \times 100 \times 30mm$ (w. baffle)	$200g$	(BLUE CANYON TECHNOLOGIES, 2015a)

Source: Elaborated by the author.

Note: c.b.: cross-boresight; a.b.: around boresight.

In the first two rows of Table 2 there are examples of star trackers developed by academic initiative. Erlank and Steyn (2014) developed on Stellenbosch University the CubeStar, a star tracker specifically designed to be used on CubeSats and built with COTS (Commercial off-the-shelf) components. Barschke et al. (2013) developed the Stella star tracker at Julius Maximilian University of Würzburg, which is part of the nanosatellite TechnoSat from the Technische Universität Berlin. These star trackers have specifications very close to those of commercial applications, with measurements made with simulations and field tests. Other three examples which were not featured in the comparison due to the absence of published data are McBryde and Lightsey (2012), developed by the University of Texas at Austin and destined to be used in CubeSats, and Rawashdeh (2013), a stellar gyroscope, which is very close to a star tracker but also provides angular velocity information, developed in the University of Kentucky and part of the KySat-2 satellite. Lindh (2014) studied the implementation of the electronic system for a star tracker, without discussing the software implementation.

For the star tracker algorithms, Erlank and Steyn (2014) used the Recursive Region Growing, Geometric Voting and Quest algorithms for centroiding, star identification and attitude determination, respectively. McBryde and Lightsey (2012) selected, in the same order, Liebe's centroid algorithm, Geometric Voting and the Singular Value Decomposition method. Star tracker software components are discussed later in detail in chapter 3.

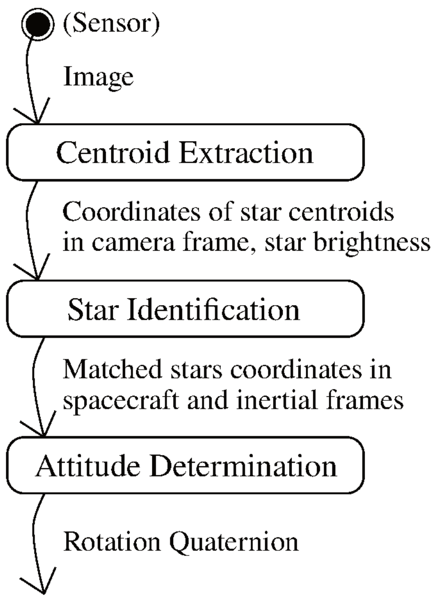
The other rows of Table 2 are commercial star trackers. In general, the available information about them is limited to the data presented in the datasheets, which are restricted to specifications that are relevant for commercialisation purposes. There are some discussions about the challenges of miniaturisation of the ST-200 of Berlin Space Technologies, which is derived from the ST-100, projected for bigger satellites (SEGERT et al., 2011). Regarding MIST, from Space Micro, there is available information about the algorithms that were used, with the internal simulation system also described (SENG et al., 2005). It was published when the technology belonged to Comtech and the star tracker was called AeroAstro MST, with the company's integration by Space Micro, which occurred in 2014. There is also information about the physical construction of the camera available for the Nano Star Tracker of Blue Canyon Technologies, which is published in Palo, Stafford, and Hoskins (2013).

The ST-16, of Sinclair Interplanetary is the commercial star tracker with the most number of related publications available. During its development phase, the project was called S3S. Enright, Sinclair, Grant, et al. (2010) discusses the necessary characteristics for attitude determination using a single star sensor as the only source of information in a satellite. The utilisation of COTS imaging sensors was discussed in Enright, Sinclair, and Fernando (2011), and results in the choice of the Aptina MT9P031 CMOS sensor (now owned by On Semiconductors) as the sensor in use in the ST-16. Later improvements in hardware and software in order to achieve better precision in attitude readings were published in Enright, Sinclair, and Dzamba (2012). The validation and tests of a Baffle were presented in Marciniak et al. (2013). Finally, Dzamba et al. (2014) discusses the validation of the ST-16 aboard a real satellite, and the improvements that were made online to the software in order to achieve the expected results.

3 STAR TRACKER SOFTWARE STACK

After an image is acquired by the image sensor, there are dedicated algorithms which are responsible for the image processing, identification of the stars present in the image and the attitude determination from this data. Thus, we can generally separate the star tracker processing computations in three parts: *centroid extraction*; *star identification*; and *static attitude determination*. Doing so facilitates the study of them as semi-independent parts and allows the recombination of individual components built using different strategies. Figure 7 shows the input and output of the composing parts of a star tracker software.

Figure 7 – Input and output of star tracker software composing parts. The boxes indicate the software components, and the arrows indicate the flux of data between them.


(Sensor)
Image
Centroid Extraction
Coordinates of star centroids in camera frame, star brightness
Star Identification
Matched stars coordinates in spacecraft and inertial frames
Attitude Determination
Rotation Quaternion

Source: Elaborated by the author.

3.1 SUBPIXEL CENTROIDING

One way to increase the accuracy of measurements is to purposefully defocus the optics so that instead of each star projecting all its photoelectrons in a single pixel of the sensor, they are distributed among several pixels. The centre of this illuminated region can then be determined with subpixel accuracy (LIEBE, 2002).

The most commonly used algorithm for determining the star subpixel centroids is the Centre of Mass algorithm, described in Liebe (2002). If we consider a region of interest window containing the pixels that form a star, we can determine the brightness DN (Equation 1) and the centroid (x_c, y_c) of the star with Equation 2 and Equation 3. In the Equations, x and y are the pixel coordinates, $image(x,y)$ is the pixel intensity value and ROI is the region of interest, delimited by the constants $ROIstart$ and $ROIend$. Thus, the process calculates the mean of the coordinate values weighted by the pixel brightness values.

$$DN = \sum_{x=ROIstart+1}^{ROIend-1} \sum_{y=ROIstart+1}^{ROIend-1} image(x,y) \tag{1}$$

$$x_c = \sum_{x=ROIstart+1}^{ROIend-1} \sum_{y=ROIstart+1}^{ROIend-1} \frac{x \cdot image(x,y)}{DN} \tag{2}$$

$$y_c = \sum_{x=ROIstart+1}^{ROIend-1} \sum_{y=ROIstart+1}^{ROIend-1} \frac{y \cdot image(x,y)}{DN} \tag{3}$$

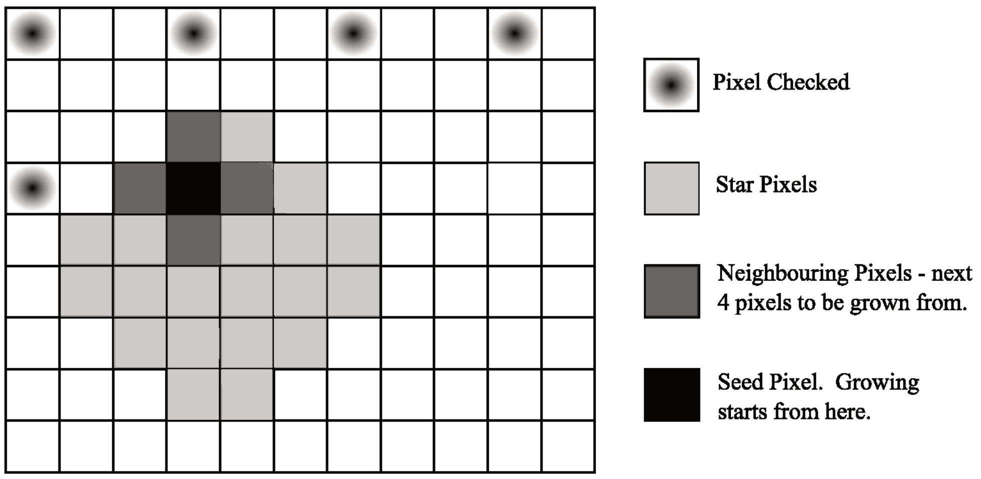
There is also a more computationally expensive method in which a Gaussian interpolation method is used to determine the centroids (QUINE et al., 2007), which theoretically should result in better estimations since the defocused projection of a star in the sensor is very close to a bidimensional Gaussian distribution. However, when compared with the centre of mass algorithm with real star images, it was found that it is less stable when successive images are taken into account (ZHANG, P. et al., 2014).

In these equations, the brightness level of each pixel is considered for weighting its importance when calculating the centroids, instead of simply considering their positions. Also, the region of interest is considered to be a square, but does not need to be so. The actual algorithm used in this work for segmenting the regions of interest of the stars instead separates the precise contour of pixels above a defined threshold. It is called Region Growing Algorithm, a recursive algorithm implemented from the description in Erlank (2013), which works as following:

1. The coordinates of a pixel (called seed) belonging to the star is given;
2. The seed is added to a list of pixels belonging to the star;
3. The seed's pixel value is set to zero to prevent if from being detected again;
4. The seed's four vertical and horizontal neighbours are checked against the detection threshold;
5. For the neighbours above the threshold, the algorithm is called recursive with that pixel as the seed;
6. The algorithm ends when no more neighbours are found above the set threshold.

Two restrictions are added to detected regions: it must exceed a given number of pixels, in order to avoid dead pixels or other types of noise; and be below a maximum number of pixels, which could indicate that big objects are being detected and possibly saturating most of the pixels of the sensor, such objects might be the Moon, the Earth or the Sun entering the FOV. The Figure 8 illustrates how seeds are given into the Region Growing Algorithm, and how the next seeds from neighbours are obtained for doing the recursion.

Figure 8 – Graphical representation of the Region Growing Algorithm, illustrating the checked pixels and how the algorithm searches for the region once a valid seed is found.


Pixel Checked
Star Pixels
Neighbouring Pixels - next 4 pixels to be grown from.
Seed Pixel. Growing starts from here.

Source: Reproduced from Erlank (2013).

The coordinates with subpixel accuracy of the centroids are then obtained from the region coordinate lists. The final output of subpixel centroiding is then x_m, y_m and the respective brightness DN, which is kept as an extra parameter. The brightness obtained from pixel values can be used directly as a feature for star identification (ZHANG, P. et al., 2014) but there are significant uncertainty in its measurements, which is the reason why many star identification algorithms do not rely on this feature. Since brighter stars have bigger regions of interest for subpixel centroiding determination, they can be consider to have less uncertainty in the measurements, which are also less likely to be a detected false star due to noise or a sensor malfunction. Some algorithms such as Grid (PADGETT; KREUTZ-DELGADO, 1997) use the brightness information to give preference to identifying the brightest stars when there are more stars detected in an image then the necessary for attitude determination.

3.2 STAR IDENTIFICATION

Taking the output of the subpixel centroiding step, a star identification algorithm is supposed to identify the sampled centroids based on an internal database of known stars. After a successful identification, both the coordinates in spacecraft-fixed frame and in the reference inertial frame will be known for multiple stars on the current FOV. This internal database is constructed according to the needs of the algorithm with information from a published star catalogue.

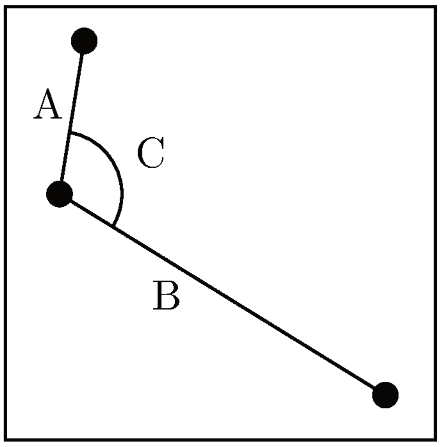
3.2.1 Literature Review

Padgett, Kreutz-Delgado, and Udomkesmalee (1997) classified star identification algorithms into two classes. The first class of algorithms treats the known stars (obtained from a star catalog) as vertices in an undirected graph G, with the angle separation between each pair of stars serving as weights for the edges. The stars detected from an image in a star sensor then forms another undirected graph G_s. The star identification problem formulation then would be defined as finding a subgraph of G that is isomorphic to G_s.

Usually, the angular distances between stars are entered in a database of polygons (the most commonly used being triangles). In some cases, brightness information is also considered for matching, but since there is a high variance when determining the star magnitude from a star sensor in comparison with location measurements, this can negatively impact performance if not carefully considered. Figure 9 shows the triangular feature, as described by Liebe (1992), which is an example of feature used in the graph isomorphism approach.

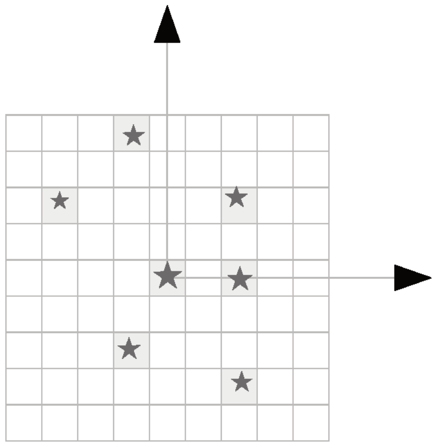
The second class of star identification algorithms approaches the problem from the perspective of pattern recognition. In this class, an identifying pattern is formed considering the neighbouring region for star components that belong to a known catalogue, and this pattern is stored in a database. Similar patterns can then be constructed from the images acquired using the star sensor. The problem formulation then becomes finding the closest pattern in the database. Examples of such patterns can be seen in Figure 9 and Figure 10.

Figure 9 – Liebe's features. Two inter-star angles A, B and an internal angle C are considered.



Source: Redrawn from Liebe (1992).

Figure 10 – The grid pattern. A binary feature is constructed based on a loose grid.



Source: Redrawn from Ho (2012).

Spratling and Mortari (2009) wrote a survey of star identification algorithms, analysing their asymptotic complexity as a form of comparison. It serves as a base for understanding the intrinsic differences between algorithms. Most of the reviewed algorithms belong to the first group, based on subgraph isomorphism, because they are by far the most commonly found in published articles. The two algorithms evaluated by the survey that belong to the second class, the grid algorithm (PADGETT; KREUTZ-DELGADO, 1997) and the radial and cyclic pattern algorithm (ZHANG; WEI; JIANG, 2008), are criticised by their worse asymptotic complexity in comparison with the members of the first class.

By doing the comparison from the perspective of noise tolerance, algorithms from the second class tend to perform better than the first class (PADGETT; KREUTZ-DELGADO; UDOMKESMALEE, 1997). Noise can manifest in star trackers in the form of smaller location accuracy of stars in the sensor plane, differences in perceived brightness (perceived stellar magnitude) or the presence of false stars. The reason for this tolerance is that working with binary patterns leave some room for changes in location of the stars while still generating the same pattern, and false stars only change a single bit, leaving the pattern still very close to the original. It is important to notice that there are today algorithms from the first class which have shown to perform very well in the presence of false stars (KOLOMENKIN et al., 2008).

Complementing this discussion, the survey of Na and Jia (2006) adds that for the same sensitivity the grid algorithm (second class) generally requires more stars in the field of view (FOV) for correct identification than other algorithms (from the first class). The advantage of using a smaller FOV angle with the same sensor resolution is that it results in higher accuracy of attitude measurements. Star patterns have richer information when multiple stars can be seen in the FOV, thus in general pattern-based star identification perform worse than subgraph isomorphism approaches that can differentiate better between small differences in angles between stars.

Ho (2012) wrote a survey that focus on star identification algorithms performance at very low FOV angles. A FOV of 4x4 degrees is considered for evaluation of algorithms. In this condition, algorithms that use information from past image frames, using tracking and combined images, are better fitted to the problem. The grid algorithm is deemed unable to work properly in this conditions, since there is a requirement for multiple stars in the FOV for correctly form the star patterns, but with the 4x4 degrees restriction, more than 70% of orientations capture less than two stars.

Guangjun Zhang (2017) considers algorithms based on star patterns to have significant advantages when compared with the traditional angular distance algorithms, due to their better tolerance to positional and magnitude noise and smaller database size requirements, but particularly criticises the grid algorithm low probability of matching a particular star, specially if it is detected close to the borders of the image sensor.

Other significant algorithms that belong to the pattern recognition class are Zhang, Wei, and Jiang (2008), which transforms the rectangular grid pattern into polar coordinates, working with the radial and cyclic features, which provided better identification rates at a cost of slower identification and Na, Zheng, and Jia (2009), which implemented elasticity in the grid algorithm, also allowing better recognition rates and less susceptibility to noise, also at the cost of increased complexity.

3.2.2 Discussion

Summarising the previous Section, pattern based star identification is less susceptible to noise, but has higher asymptotic complexity and is more limited when working with smaller

FOV angles. Since it usually works with binary patterns, it is also relatively simple to make a hardware implementation as a digital circuit in configurable hardware. For the grid algorithm, the identification can be preformed by calculating scores using the Hamming weight (sum of ones in the binary word) of the resulting Boolean bitwise AND operation between the sensor obtained pattern and the database patterns, with the highest score indicating that the database entry is the closest to the sensor pattern. While the original implementation still has worse algorithm complexity, the internal database of patterns can be partitioned and the comparisons can be made in parallel on multi-core architectures without modifying the results of the algorithm.

Here, another example of pattern recognition problem is presented in order to form a parallel with the star identification problem. The matching of *Local Features* is defined as image patterns that differ from their neighbourhood in terms of intensity, colour and texture. *Local Features* can be small image patches, edges or points, and in modern publications are also called interest points, interest regions or keypoints. Good *Local Feature* patterns are repeatable, meaning that the same scene being captured in different conditions of point of view or illumination still detect a high percentage of similar features. They should also be distinctive, to avoid having multiple matches when comparing the patterns leading to erroneous pairings (SIEGWART; NOURBAKHSH; SCARAMUZZA, 2011, p. 212–213).

In this parallel context of *Local Features*, binary descriptors were shown to have several advantages over the more established vector-based descriptors. They are compared using Hamming distance, which can be computed by performing a bitwise XOR operation followed by determining the Hamming weight (counting the ones of the result). This operation can be performed quickly on modern computers, where there are dedicated instructions for bit counting (MUJA; LOWE, 2012). These instructions are POPCNT on Intel x86 architectures which support SSE4 instructions (INTEL CORPORATION, 2007), and VCNT on ARM architectures with NEON instructions (ARM LIMITED, 2013).

In one hand, it is desirable to maintain the natural advantage points of pattern recognition star identification. For example, better noise tolerance and the use of efficient operations when being run in modern hardware are strong points originally present in algorithms of this group. On the other hand, it is worthwhile to consider improvements in asymptotic complexity of the database search and in better detection rates when fewer stars are present on the FOV.

With such developments, pattern recognition algorithms could become more advantageous to use in miniaturised star trackers, specially when considering working with binary descriptors in current microprocessors or FPGA’s. Fewer instructions or clock cycles would be necessary to perform the same computations, without compromising the rate of successful attitude determination. This work is an attempt to address these limitations of pattern recognition based star identification algorithms using binary descriptors in order to make them more suitable for use in state of the art star trackers.

3.2.3 Star Catalog

Star catalogues contain astrometric and photometric data, usually collected by observations of specialised satellites. Such data, in the context of star simulators and star identification algorithms, is used as the fundamental data for constructing the internal database of stars. In this work, the Hipparcos[1] star catalogue (ESA, 1997) was selected for use, due to its highly accurate data for brighter stars. Current CCD and CMOS sensors are able to detect stars around magnitude 6.0 and lower[2] (ENRIGHT; SINCLAIR; FERNANDO, 2011), making the selected catalogue appropriate. The Hipparcos-2[3] catalogue (LEEUWEN, 2007) was a later improvement on the accuracy of original Hipparcos data, achieved by a new reduction of astrometric data. The updated data was considered in this work, complemented with information from the original Hipparcos catalogue when such data was not available on the updated version.

The entries of interest in Hipparcos and Hipparcos-2 catalogues are shown in Table 3. On the Table, the most important fields are RArad, DErad and Vmag. They correspond, respectively, to the right ascension coordinate, the declination coordinate, and the visual magnitude (brightness) of the stars. The coordinates are represented in the International Celestial Reference System (ICRS), in the epoch 1991.25, defining the inertial reference system of the catalogue (as defined in section 2.1). The Hipparcos identifiers are useful for pairing entries from the two catalogues. The proximity and coarse variability flags indicate stars that have close neighbours in their immediate proximity and stars that have considerable variability in magnitude, respectively. These flags are useful to determine stars which are bad candidates for being used as references. Very close stars might be mistaken as a single brighter star, or a star that show significant variability can produce spurious detection on the limits of sensitivity of the hardware. Finally, proper motion information are used to correct for the perceived movement of stars through time in the celestial sphere. Proper motion information can be used to correct for the perceived movement of stars through time in the celestial sphere.

Both catalogues are presented as ASCII text files, with fields being always delimited by the same length in bytes. The fields were parsed in the software. The entries shown in Table 3 were selected for building two tables of data, one for each catalogue. These two tables were then joined in a single table by matching their HIP identifiers. Hipparcos have more entries of stars than Hipparcos-2. These extra entries were deemed as unreliable and excluded from the joined table.

The entries of the catalogue which are above the threshold of magnitude considered, and thus deemed to be undetectable by the sensor, were eliminated. This greatly reduced the dimensions of the database constructed from the data. The star coordinates were updated to current time through proper motion correction, then transformed from their angular representation (right ascension α and declination δ) into a unit vector (inertial) in Cartesian coordinates. The process

[1] `http://cdsarc.u-strasbg.fr/viz-bin/Cat?I/239`

[2] Counter-intuitively, lower values of star magnitude actually mean brighter stars.

[3] `http://cdsarc.u-strasbg.fr/viz-bin/Cat?I/311`

Table 3 – Entries from Hipparcos and Hipparcos-2 which are of interest.

Symbol	*Catalogue*	*Label*	*Description*	*Unit*
-	Hipparcos	HIP	Identifier (HIP number)	-
-	Hipparcos	Vmag	Magnitude in Johnson V	mag
-	Hipparcos-2	HIP	Hipparcos identifier	-
α	Hipparcos-2	RArad	Right Ascension, ICRS, 1991.25	rad
δ	Hipparcos-2	DErad	Declination, ICRS, 1991.25	rad
$\mu_{\alpha}*$	Hipparcos-2	pmRA	Proper motion in Right Ascension	mas/year
μ_{δ}	Hipparcos-2	pmDE	Proper motion in Declination	mas/year

Source: Adapted from ESA (1997) and Leeuwen (2007).

is detailed in sequence.

Proper motion is defined as the time derivative of the positional coordinates of right ascension (α) and declination (δ), as seen in Equation 4.

$$\mu_{\alpha} = \frac{d\alpha}{dt}, \quad \mu_{\delta} = \frac{d\delta}{dt} \tag{4}$$

The Hipparcos entries are pmRA ($\mu_{\alpha}*$) and pmDE (μ_{δ}). The asterisk in the proper motion of right ascension denotes that it is converted to great circle measurements for being directly comparable to μ_{δ}. It is necessary to undo this conversion before applying the corrections (ESA, 1997, p. 25), by determining μ_{α} from Equation 5.

$$\mu_{\alpha}* = \mu_{\alpha} \cdot \cos(\delta) \tag{5}$$

Before making the correction, the units for the proper motion variables must be converted from milliseconds of arc/year to radians/year, as shown in Equation 6 and Equation 7.

$$\mu_{\alpha}(rad/year) = \frac{\mu_{\alpha}(mas/year) \cdot \pi}{3600 \cdot 1000 \cdot 180} \tag{6}$$

$$\mu_{\delta}(rad/year) = \frac{\mu_{\delta}(mas/year) \cdot \pi}{3600 \cdot 1000 \cdot 180} \tag{7}$$

Then it is possible to correct the proper motion by using its definition (Equation 4), which is represented in Equation 8. Here the time t unit is years.

$$\alpha = \alpha + \mu_{\alpha} \cdot t, \quad \delta = \delta + \mu_{\delta} \cdot t \tag{8}$$

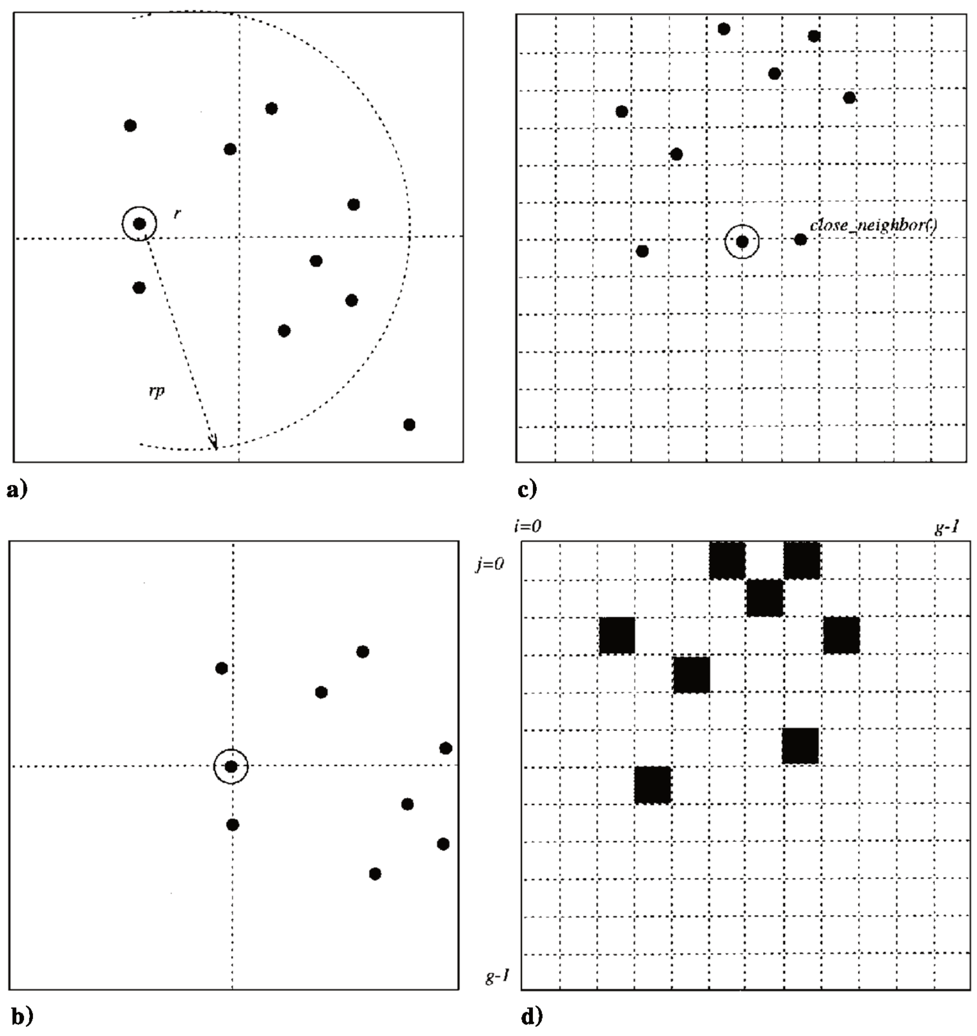
The new α and δ coordinates can then be transformed into unit vectors using Equation 9.

$$uv = \begin{bmatrix} \cos(\alpha) \cdot \cos(\delta) \\ \sin(\alpha) \cdot \cos(\delta) \\ \sin(\delta) \end{bmatrix} \tag{9}$$

3.2.4 The Grid Algorithm

The Grid Algorithm, proposed by Padgett and Kreutz-Delgado (1997), is the most widely known algorithm for star identification problem that works from a pattern recognition perspective. The idea behind it is to describe each of the known stars based on their surrounding neighboring stars with a (preferably) unique pattern and create a database of these descriptors. This process is done offline. A similar pattern is then extracted online with the same process for each star that belongs to the image acquired by the optical sensor. A distance criterion is then applied, and the descriptor with minimal distance from the catalogue is assumed to be the correct match. In this Section, a summary of the algorithm is presented with the concepts described as originally written in Padgett and Kreutz-Delgado (1997).

Figure 11 – Construction of the binary grid pattern.



Source: Reproduced from Padgett, Kreutz-Delgado, and Udomkesmalee (1997).

In the grid algorithm a star pattern is formed from the stars by following these steps, which are matched to the visual representation in Figure 11:

1. A star r is selected as the reference to create the pattern;
2. The r star and a part of the surrounding sky with radius pr is translated to the center;
3. A loose square grid of side g is placed, with the pattern rotated so that the closest neighbour star will lie in the x coordinate axis (achieving rotation invariance);
4. A g^2 length bit vector is derived from the grid pattern. The presence of a star in a cell is represented linearly in the vector so that the bit $k \cdot g + i$ is 1, while its absence is represented as the bit 0.

For example, for extracting the pattern of Figure 11, working with $g = 12$, and considering the upper left cell as the origin coordinate $[0,0]$ as being represented by the least significant bit (LSB) and working left to right, up do down, the pattern on Figure 11 is represented, in hexadecimal notation, as: $0x00000000000000000000808000001010404 00a0$, where the pixels $\{5,7,18,26,32,40,67,75\}$ are set.

More formally, considering that i is the set of database stars, and j is the set of stars in the sensor image, and that pat_i is the set of patterns from the database and pat_j is the patterns of the currently selected reference star to create the pattern, the maximum score needs to be found (Equation 10), with the match function being defined as Equation 11. The & operator represents the logical AND operation.

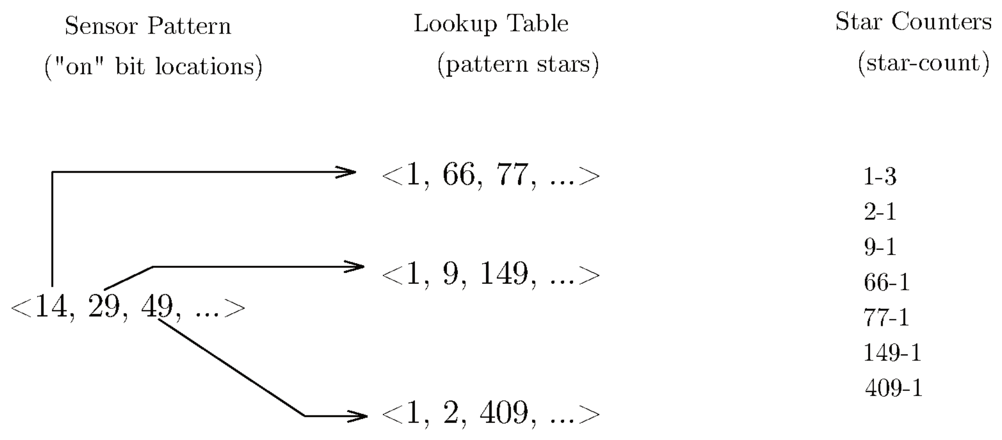
$$max_i(match(pat_j, pat_i)) \tag{10}$$

$$match(pat_j, pat_i) = \sum_{k=0}^{g^2-1} (pat_j(k) \ \& \ pat_i(k)) \tag{11}$$

The construction of the internal database requires the introduction of a few restrictions. From the visible stars (V), some unreliable stars are excluded from becoming entries in the database: stars that have variable brightness; binary stars; or other configurations with high proximity. The remaining stars that form the database entries are called the Reference Stars R. For each member of R, which is a reference star, the stars within the pattern radius pr of this reference star that are members of V are then used to form the pattern, using the procedure previously mentioned for building the star patterns. Members of the catalogue which have fewer stars than a constant threshold value α, the known star density, are excluded from the catalogue. α will restrict the number of stars in the sensor image that are required to result in an unambiguous match, meaning its value is at least two. This will determine the best possible recognition rate of the algorithm.

A different way of representing the database was proposed in the original article in order to reduce its memory requirements, which also resulted in a computationally superior version of the algorithm. In this representation, a lookup table (LUT) is created (visually represented in Figure 12). The possible cell numbers k from the pattern (with bit positions 0 to $g^2 - 1$) serve as the table indices. For each star *i* from the reference stars *R* that have a 1 in the cell *k*, its identifier *i* is added at the table at the line with index *k*. This means that each line *l* in the table contains a list of the identifiers of the stars which have a 1 in the cell *l* of the original pattern. When a pattern is acquired from the sensor, the best score is found by scanning the lines at the lookup table with indices equal to the active cells of the sensor pattern (with value 1). A counter is incremented for each star identifier listed in the scanned lines. At the end of the process, the highest value counter indicates the star that has the best score. This procedure is mathematical equivalent to the one described by Equation 10 and Equation 11, with a database that consumes less memory[4]. The best match is always returned, even if the absolute scores are low.

Figure 12 – Grid database in the transposed lookup table format. The lookup table has a number of lines equal to the number of bits of the descriptor. For each "on" bit on the sensor pattern, the corresponding line of the lookup table is looked up. It contains all the star identifiers which also have a "on" bit in the same position. By counting the frequency from which each star is observed in the LUT entries, adding up the star counters, it is possible to find the identifier of the most common star and the its score. The result is mathematically equivalent to Equation 10 and Equation 11.



Source: Redrawn from (PADGETT; KREUTZ-DELGADO, 1997).

Once the matching process is complete, a clustering process is applied to the list of identified stars. Stars are clustered together with an area that is limited to a single FOV. In this process, the largest cluster of stars that are found to be closer than the maximum FOV angle are considered to be correct, thus removing wrong matches that fall outside of this cluster area. The clustering procedure is defined, but the actual cluster algorithm that was used is not specified. A relatively simple algorithm called Leader Clustering was used for this purpose in this work's implementation (HARTIGAN; HARTIGAN, 1975, p. 75).

[4] This happens due to most cells being 0 values in the patterns, meaning that there are more empty space than stars in the sensor images. It is assumed that g^2 is much greater than the average number of stars per pattern.

The algorithm works by (1) selecting the first star of the list as the leader of the cluster and removing it from the list, then (2) assigning all other stars within the FOV angle distance to this cluster and removing them from the list. (3) Step 1 is then applied recursively to the remaining stars in the list, until all stars are members of a cluster. The largest cluster is selected as the output of the verification routine. In case of two or more clusters being the largest, an error is signalized and there is no output.

This algorithm requires a single pass through the identified stars list to do the clustering, thus being very fast. A trade-off for the fast speed is that it can produce different results depending on the ordering of the input list, as it assigns the stars immediately to the current cluster being considered if it satisfies the FOV angle distance in which the angle is within the possible FOV, disregarding cases where a star could also be assigned to a different cluster, perhaps within a smaller angle of the reference star. There is also no careful selection of the cluster leaders. With this in mind, the diagonal of the sensor is considered as the largest angle permitted for stars to be considered as within the same cluster.

3.2.5 Discussion of the Grid Algorithm

One of the defining characteristics in the grid algorithm is that rotation invariance is achieved through aligning the closest neighbour into the image's x coordinate axis. If a mistake is done when selecting the next neighbour for creating the pattern, due to sensor noise for example, the complete pattern is compromised. This can be seen as a disadvantage of the algorithm, but this particularity can also be explored in a positive way. It was used for improving the database search procedure, as it will be discussed later in subsection 5.2.3.

The restructuring of the database in the lookup table format brought benefits in memory requirements and computationally requires less resources, since only the active cells are considered for the database search. However, this kept it from exploring the advantages of using a binary descriptor and doing boolean comparisons (which may benefit from dedicated hardware instructions). Bringing back the parallel with *Local Feature* descriptors (as discussed in section 3.2). This rearrange of the database in LUT format was also adopted by later algorithms, inspired by the grid algorithm (ZHANG; WEI; JIANG, 2008; ZHAO et al., 2017).

There is also one important difference from *Local Feature* descriptors: the grid algorithm score function (Equation 11) works with the boolean AND operator; while in *Local Feature* descriptors this score uses the boolean XOR operation.

For the sake of understanding the selection of the AND function, a naïve approach to derive a scoring function would be to compare the set and unset bits from the database and sensor and whether it is expected for the score to go up or down, as show in Table 4. In this context, it would seem as the NXOR boolean operation would be more suitable to represent the behavior of what would be expected of a scoring function, rather than the AND function that was originally selected. The operator was probably chosen instead for the sake of simplicity, a decision which can be understood with a more detailed analysis of the scoring process.

Table 4 – Classification of the scores and comparison with boolean functions (idealized).

Database	Sensor	Classification	Scoring Behavior	AND	NXOR
0	0	true negative	up	0	1
0	1	false positive	down	0	0
1	0	false negative	down	0	0
1	1	true positive	up	1	1

Source: Elaborated by the author.

For all the patterns contained within the database, constructed from the star catalogue, a set bit means that one or more stars are present within the pattern cell and, complementary, a zero value means that no star is present in the region delimited by the cell. However, the same cannot be said for the patterns derived from the sensor image. The translation and rotation steps that are part of the pattern construction make out-of-image areas fall within the space considered for the construction of the pattern. This means that zeros can both mean that there are no stars within the cell, or that the existence of stars is unknown because the information is outside of the boundaries of the sensor.

It can be concluded that from the original way that the patterns were constructed, only the set values were reliable for the matching function comparison. Therefore only the AND boolean operation would make sense for the scoring function.

The downside of this approach is that only the true positives are considered for scoring. Neither false positives nor false negatives impact in score reduction. True negatives also do not contribute for a higher score. The representation of the unknown state as a zero in the pattern prohibits the use of the NXOR operator, which ideally would consider all the possible comparison cases in scoring.

3.3 STATIC ATTITUDE DETERMINATION

Attitude determination can be differentiated from attitude estimation in whether they are a static method which only information that is acquired at the same time (or very close) is considered, or when previous measurements are also considered in a dynamic system. In the first case, there is no memory of previous attitude measurements used when computing the current attitude. In the second, previous attitude information can be also considered when computing the new attitude estimate (MARKLEY; CRASSIDIS, 2014).

For dynamic attitude estimation, a Multiplicative Extended Kalman Filter (MEKF) (MARKLEY, 2003; TRAWNY; ROUMELIOTIS, 2005) can be used to simultaneously estimate the attitude in quaternion representation and correct gyroscope drift with observed attitude from a star tracker. It differs from the Extended Kalman Filter (EKF) in the prediction step, where the control information is considered as a quaternion multiplication instead of a sum, avoiding singularities in the covariance matrix problems for the estimation of attitude in quaternion form. Using a quaternion representation for attitude determination has a clear advantage over using rotation matrices with Euler angles because for three dimensional rotations there is no need to take precautions against gimbal lock problems.

In this work only a static attitude determination algorithm was implemented, since it is enough for verifying the complete star tracker structure. Static attitude determination is also a pre-requisite for attitude estimations, which require attitude measurements for correcting the filter state vector estimations.

The generalised problem of determining the attitude in the form of a rotation matrix between two sets of n points which best minimises the least squares error, between the first set is rotated by the matrix and the second set, was presented by Wahba (1965). This essentially is the problem that must be solved for finding an attitude on a star tracker, where the rotation between the unit vector points in the inertial frame (from the star catalogue) and the unit vector of the detected stars by the sensor must be found with the minimum possible error. There are many algorithms which solve Wahba's problem, with many of them using the quaternion representation for attitude instead of a rotation matrix form. A discussion of these algorithms can be found in Markley and Crassidis (2014).

For fulfilling the objective of verifying the star identification algorithm, the Quaternion Estimator (QUEST) algorithm (SHUSTER; OH, 1981) was implemented, based on the equations show in Hall (2003). It is the most widely used attitude determination algorithm for solving Wahba's problem, with a long history of successful application and has some simplifications for reducing the computations. One problem of the QUEST algorithm is that in its original form it shows singularity problems in the computed matrices when the rotation angles approach 180 degrees, which resembles gimble lock problems with Euler angles for rotation matrices, a problem that can be solved by the Method of Sequential Rotations (MARKLEY; CRASSIDIS, 2014).

4 VERIFICATION PLATFORM AND STAR SIMULATOR

In order to ensure the correctness of the star tracker algorithms during implementation, and to provide metrics from which the optimized versions of the algorithms could be compared to the references, it became necessary to implement a structure for verification.

During the research of metrics that could be applied to the specific implementation of this work, it became clear that there is no current standard tests from which star trackers, or more specifically star identification algorithms, could be compared to each other. The metrics employed by different authors vary[1], and naturally have an inclination to focus on the specific contributions involved.

Kolomenkin et al. (2008) expresses his concern with this point, “Many authors have referred to different aspects of star tracker performance, such as speed, accuracy, memory requirements, and stability. But each of them used a different configuration.” Many authors have done small comparatives of their algorithms with existing references which were relevant at the time of publishing. These comparatives usually involve one or two other algorithms which were implemented for the sake of the examination. Most of the tests require a star simulator in order to be realized, but no specialized simulator is shared within the scientific community.

This approach has been successful in enabling the showcase of the author’s contributions, but have a few downsides:

1. The implementation of the reference algorithms and simulation environments might differ and taint the results of the comparisons involved;

2. In many cases it results in rework. By reducing the time implementing reference algorithms and creating testing platforms with sky simulators, this resource could be reallocated to the proper research, potentially improving the quality of the scientific work;

3. Individual testing platforms often produce different metrics as outputs, which in many cases cannot be directly compared, which reinforce the need to redo the comparisons, presenting positive feedback for the two previous items.

Therefore, instead of building a testing platform that would only target the algorithms involved in this work, a more general testing platform that can target other present and future algorithms for star trackers was considered.

The general structure of the platform is based on black-box testing. The internal mechanism of the star tracker sub-components were ignored, and the testing procedure focused on the outputs generated in response to controlled inputs and execution conditions (GAO; TSAO; WU, 2003, p. 119–120).

With a proposed standard for the input and outputs, the algorithms being tested could be interchanged with no necessary modifications to the testing platform. The platform would

[1] The references associating authors to specific metrics are shown later in section 4.4.

be composed of a star sky simulator, with controllable addition of noise in various forms, the testing procedures and the generation of statistics and graphics as an output, in a standard format permitting direct future comparisons. More specific tests that are required for a given algorithm for other reasons can be implemented as extensions to the testing platform, benefiting from the existing structure.

4.1 RELATED WORK

In this section, a review of related work on the subject of star simulators and generation of synthetic sky images is presented. This is important to generate the input for star tracker testing. Similar applications of the UVM structure to computer vision systems are also discussed in the context of this work.

4.1.1 Star Simulators and Testing of Star Trackers

On the subject of star image synthesis, a mathematical model is presented in Hua-Ming, Hao, and Hai-Yong (2015) and Guangjun Zhang (2017). Both works use Gaussian functions as the Point Spread Function (PSF), for producing slightly defocused projections of stars, and for simulating random background noise. The approach and equations used in the present work for image synthesis and noise modelling are also similar, but the rotations performed during the generation of the star images were done using quaternions, instead of Euler's rotation matrices. The advantage of the former attitude representation is in the fact that it is not subject to gimbal lock limitations.

A more thorough model of the optical system of a star tracker, which includes the lens and image sensor, can be seen in Knutson (2012). The simulator, written in MATLAB/Simulink, considers the physical aspects of the optical system for emulating noise.

Some simulators also consider the dynamic conditions of the star tracker during image synthesis. For example, in Wang et al. (2012) a mathematical model is presented for generating smeared images of a star field. This covers the simulation of some possible scenarios, such as a star tracker that is attached to a spacecraft with high angular velocity, generating sky pictures with motion blur during the exposure time of the photograph. Practical implementations of simulators with such characteristics can be seen in De Brum et al. (2013) and Filipe et al. (2017), which can test star trackers in dynamic conditions.

Another relevant feature is the simulator's ability of working in hardware-in-the-loop configurations, either in open or closed loop. Examples can be seen in Filipe et al. (2017) and Samaan, Steffes, and Theil (2011). The former work also presented a miniaturized test bench where the test equipment rotates with the device under test (DUT), reproducing actual change in attitude. Hardware-in-the-loop testing is also possible with the structure presented for the verification platform in the current work. The main difference is the focus. Our platform was built to be used also during the early stages of development, before hardware implementation.

Within the mentioned simulators, the main language used for programming was MATLAB, followed by C++. Some examples of the latter are Filipe et al. (2017) and Ying Zhang et al. (2011). One advantage that these simulators have is that they support the generation of images using OpenGL for hardware acceleration, speeding up the synthesis of individual images. As our verification platform is developed in the same language, this feature can be added in future instances.

Of all the previously mentioned simulators, none are available online for download, nor published as free software. The free and open source software Stellarium[2], which is intended for users as a planetarium, is sometimes also used for testing star trackers (MCBRYDE; LIGHTSEY, 2012). Since it is being used outside of its main purpose it presents some limitations, such as difficulties to add controllable noise or for doing simulations in closed loop configurations.

In contrast to the presented solutions, our simulator is implemented with the specific features for testing star tracker algorithms in mind, and, at the same time, built from ground up with a focus in reusability. Our solution is unique in the sense that it becomes the sequence generator component of a UVM structure, which standardizes the way the test cases are built. The standard package presented in subsection 4.3.2 simplifies the interfaces between the components under the test and the UVM test bench in a black box construction, so that it becomes simple to interface star tracker algorithms to the developed verification environment.

Table 5 shows a summary of the presented star simulators based on information published, as none of them are available online, and neither are published as free software.

Table 5 – Instances of star simulators in software.

Reference	Language	PSF	Noise Model	Attitude Repr.
Fialho and Saotome (2005)	C	Gaussian	Gaussian	Euler Matrices
Samaan, Steffes, and Theil (2011)	MATLAB	Gaussian	Not described	Quaternion
Knutson (2012)	MATLAB	Gaussian	Multiple	Quaternion
De Brum et al. (2013)	C	Gaussian	Gaussian	Euler Matrices
Erlank (2013)	MATLAB	Gaussian	None	Quaternions
Hua-Ming, Hao, and Hai-Yong (2015)	C++	Gaussian	Gaussian	Euler Matrices
Guangjun Zhang (2017)	MATLAB	Gaussian	Gaussian	Euler Matrices

Source: Elaborated by the author.

4.2 SYSTEM LEVEL VERIFICATION

The demands of miniaturisation of star trackers as part of aerospace systems implies that today they use embedded design schemes with high integration levels for minimising the weight, size and power consumption. Thus, the involved algorithms usually run in embedded

[2] `https://stellarium.org/`

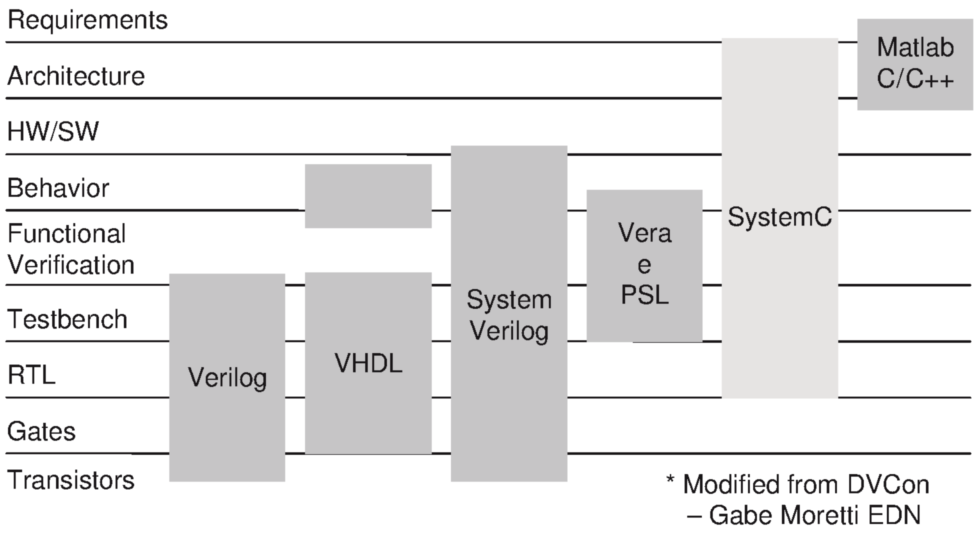
processors, for example RISC (Reduced Instruction Set Computer) processors based on ARM architecture (ZHANG, G., 2017). There is also the possibility of implementing the algorithms in FPGA devices directly in digital circuit format (ZHAO et al., 2017). Thus, it makes sense to implement verification platforms at the system level, considering that existing software[3] could also be co-verified with the hardware counterparts, either simulated or as hardware-in-the-loop approaches.

4.2.1 SystemC

SystemC is a language[4] for ESL (Electronic System-level Design) which supports multiple levels of abstraction (Figure 13), thus permiting the simulation of high level abstraction such as running the code in the host modeling processor (called Direct Execution), but also the simulation of software running in an Instruction Set Simulator (ISS), down to the Register Transfer Level (RTL) (BLACK et al., 2009, p.13).

The SystemC language was introduced in 2005 and standardized in 2012 (IEEE, 2012). In addition to SystemC, resources such as Transaction Level Modeling (TLM), verification libraries, Analog Mixed-Signal (AMS) and Testbench implementation tools form a very useful set for modeling, simulating and evaluating Hardware and Software for complete systems (RADPOUR; SAYEDI, 2018). SystemC can be used for high level synthesis, but only a subset of the language is supported for this purpose (OPEN SYSTEMC INITIATIVE et al., 2009).

Figure 13 – Levels of abstraction covered by languages.



Source: Reproduced from Black et al. (2009, p.15).

3 Which in this case could be also understood as firmware.

4 SystemC is not a standalone language, but a library which extends the C++ language.

4.2.2 Universal Verification Methodology - UVM

The verification procedures are important in the design flow of electronic systems to reveal the existence of potential faults in the design, which can later be fixed. SystemC supports the use of the Universal Verification Methodology (UVM), which is an IEEE standard (IEEE, 2017) that has been suggested for use for verification procedures, introduced in 2015 and standardized by the Accelera Systems Initiative (ACCELERA SYSTEMS INITIATIVE, 2015).

The UVM standard is available in practice in two Hardware Description Languages (HDLs): SystemVerilog and SystemC, being initially applied to the first language, then ported to the second.

The main advantage of the specification is that a testbench with the proposed structure is composed of reusable Universal Verification Components (UVCs) following a consistent architecture, which are kept separate from the Device Under Test (DUT) (HEIGHT, 2013). In other words, this block architecture facilitates the reutilisation of testing components, enforcing a modular structure through a set of rules.

4.2.3 Verification of Computer Vision Systems with UVM

There are existing publications, in other areas of computer vision applications, where the UVM-SystemC verification environment was used and paired with OpenCV on the earlier steps of development. The similarities between them and the present work support the idea that the methodology can successfully be applied for the design of star trackers, which can be understood as computer vision systems.

In Mefenza, Yonga, and Bobda (2014), a verification environment using SystemC and UVM was created for computational demanding video-based embedded systems. A system design starting with an executable specification in C++ and OpenCV was verified, and the system refined into lower levels of abstraction as an FPGA based smart camera, where the RTL portion initially modelled in SystemC was ported to VHDL. The final system works with a Zynq-7000 SoC, with software running in the dual core ARM processor and hardware in the FPGA portion.

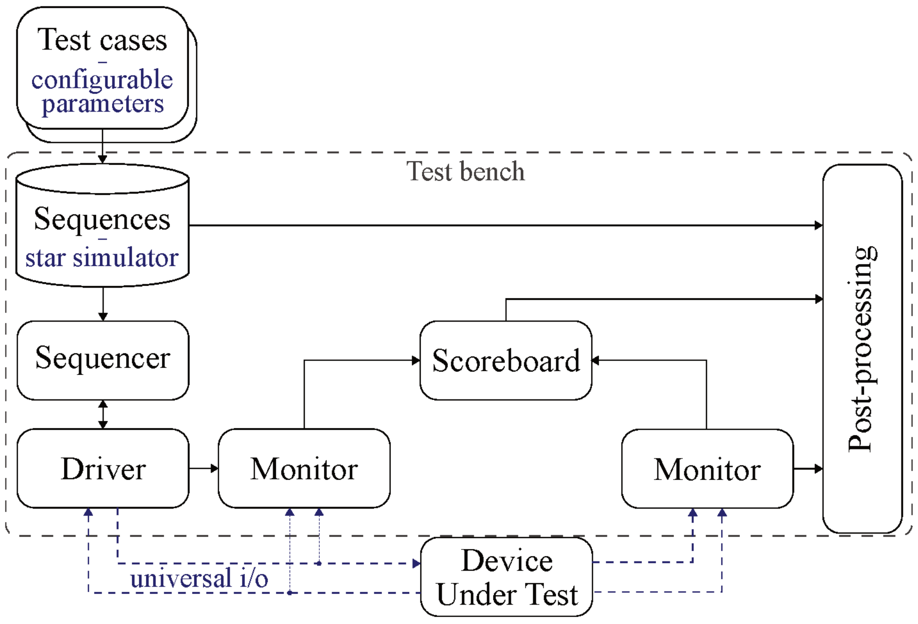
Similarly, in Campos et al. (2017) a UVM-SystemC environment was used to build a framework for the design and validation of face detection systems. Differently, no hybrid system is considered, but instead, a high-level model developed using OpenCV is used as the golden reference model, from which complete hardware implementations of the face detection algorithms can be compared during the tests. Instead of VHDL, the language used for hardware description was SystemVerilog.

4.2.4 Structure of the Verification Platform

In this section, the overall structure of the verification platform is explained. The platform follows the structure and terminology of UVM-SystemC (BARNASCONI et al., 2014).

Figure 14 links UVM components with their specialised functionality needed for the verification of star trackers.

Figure 14 – Structure of the verification platform. The UVM nomenclature for blocks is linked to their specialised functionalities for the particular case of verifying star trackers.



Source: Elaborated by the author.

The main function of the *test bench* in UVM consists of the interaction with the *Device Under Test* (DUT), through injection and data collection on the created interfaces, according to the desired *test cases*. The *test bench* components are hierarchically separated, and they assume different tasks.

The creation of test data is performed at a high-level of abstraction through the aid of C++ features, mathematical functions, and libraries. This feature of UVM-SystemC allowed the creation of a complete *star simulator* (section 4.3) to generate input for either the entire DUT or its separate components. In our implementation, the star simulator corresponds to the *sequences* component, and the *configurable parameters* (subsection 4.3.3) define the *test cases*.

The data generated by the simulator are then delivered to the *driver* component. It will receive the items, translating or adapting them for the DUT interface. Furthermore, the *input* and *output monitors* behave like testing probes, accessing the DUT interface and capturing relevant information.

The *scoreboard* component is then responsible for comparing the outputs of the system with the reference, effectively producing scores, which are post-processed as required. The *post-processing* consists, for example, of analysing the scores produced by UVM for generating tables and plotting graphics. In the case of the current implementation, the scoreboard data is written to a file. The data is then automatically processed by an auxiliary program written in

Python. The plots displayed later on chapter 5 are examples of the post-processing.

The verification platform is very flexible in generating data and testing different DUTs. Such flexibility was possible due to the creation of a universal data packet. The packet encapsulates the *universal i/o* structures, generated by the *sequences* block, and presented in subsection 4.3.2. The packets are then transmitted to the interfaces of the DUT. They contain input information that can be used by the individual components (Figure 7), whether working together as a whole or separately.

The transportation of these packets to the DUT was also standardised with the TLM 2.0 communication standard. This abstract communication pattern transports the data without the need for further detail on how the data would be transported in the real system.

Two distinct DUT wrappers were developed by us. The first interfaces the TLM packets from C++ structures to VHDL, and the second transports the structures through TCP/IP interfaces. They were used to demonstrate the verification of VHDL components for FPGA designs and hardware-in-the-loop designs, respectively.

As all the components are implemented with the object-oriented paradigm, they can easily be reused between tests to the extent it makes sense to. This is made possible by the inheritance and polymorphism features of the C++ language. The UVM structure enforces the organisation of the test bench in such a way that it becomes progressively more straightforward to implement new test cases, as most structures can be reused.

By observing the complete UVM structure, it is possible to conclude that this environment is built with modularity in mind, supported by its basic components. The stimuli are generated in high abstraction level, and are progressively refined until they reach the DUT. The accumulation of results, reversely, progresses from the lower levels to the higher.

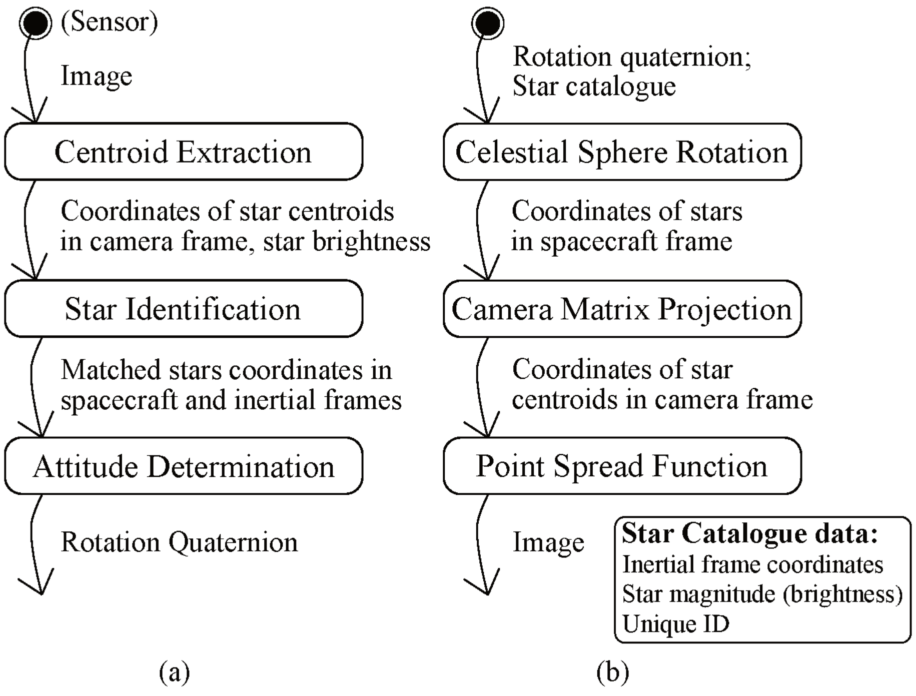
The base element of the UVM test bench is the Transaction Level Modelling (TLM). These transactions are communications between functional blocks and encapsulate data that can be information packages, for example the address and data in a serial communication protocol, without exactly containing the information of how this data would be delivered to the DUT. Within the analogue domain, packages can contain data for configuring a sinusoidal power source, sawtooth, etc. This allows the generation of input data with the freedom of working in higher levels of abstraction, without having to worry with the details of the signal level (RATH; ESEN; ECKER, 2014).

Thus, the UVM is well structured to serve as the fundamental base for implementing the proposed testing platform. It enables the testing platform to follow a well documented structure, the easy implementation of the proposed tests and can be expanded and extended easily for the implementation of future tests by the community. By pairing the UVM standard with SystemC, it is also possible to apply the same testing platform to different DUTs, and support both software, hardware and mixed environments in co-verification.

4.3 STAR SIMULATOR

The main building blocks of a star tracker and a star simulator can be seen in Figure 15. Note that the basic flow of the star tracker (a) has a dual structure when compared to the star simulator structure (b), working in reverse order. The exception is that some of the data is provided directly by the star catalogue.

Figure 15 – (a) Star tracker software input and output; and (b) star simulator stages. The boxes indicate the software components, and the arrows indicate the data flow between them.


(Sensor)
Image
Centroid Extraction
Coordinates of star centroids in camera frame, star brightness
Star Identification
Matched stars coordinates in spacecraft and inertial frames
Attitude Determination
Rotation Quaternion
(a)
Rotation quaternion; Star catalogue
Celestial Sphere Rotation
Coordinates of stars in spacecraft frame
Camera Matrix Projection
Coordinates of star centroids in camera frame
Point Spread Function
Image
Star Catalogue data:
Inertial frame coordinates
Star magnitude (brightness)
Unique ID
(b)

Source: Elaborated by the author.

In essence, the mathematical models for image synthesis for star simulators mentioned earlier in subsection 4.1.1 work basically with the same principles. The model followed by us also shares the same basic structure, with the main difference being that quaternions are used for performing the required rotations instead of Euler's rotation matrices. While all the composing equations can be found in literature, this section aims to consolidate them and explain how we connected them to each other to create a working simulator.

The starting point is the star catalogue, in the case of this work, Hipparcos-2 is used (LEEUWEN, 2007). Initially, all the steps mentioned in subsection 3.2.3 are applied to the catalogue, with the result being a celestial sphere composed by unit vectors of the stars above the magnitude threshold that was considered.

4.3.1 Generating a Synthetic Star Image

After all the steps mentioned in subsection 3.2.3 are applied to the catalogue, the result will be a celestial sphere composed of unit vectors. All stars above the magnitude threshold being considered will have a respective unit vector on the sphere.

The desired celestial sphere attitude is expressed in the form of a rotation quaternion, which can be known or randomly generated. Using quaternion multiplication, the original attitude of the reference system can be rotated into the desired attitude. Following the notation on Markley and Crassidis (2014), Equation 12 shows the form of the rotation quaternion **q**, and how it is constructed from a unit vector **u**, representing the desired axis of rotation, and the desired angle of rotation θ.

$$\mathbf{q} = \begin{bmatrix} \mathbf{q}_{1:3} \\ q_4 \end{bmatrix} = \begin{bmatrix} \sin(\theta/2) \cdot \mathbf{u}_{1:3} \\ \cos(\theta/2) \end{bmatrix} \tag{12}$$

A 3D unit vector **v** (of a star) can be expressed in quaternion form according to Equation 14. The rotation itself is performed using Equation 14.

$$\mathbf{p} = \begin{bmatrix} \mathbf{v}_{1:3} \\ 0 \end{bmatrix} \tag{13}$$

$$\mathbf{p}' = \mathbf{q} \otimes \mathbf{p} \otimes \mathbf{q}^* \tag{14}$$

The superscript $*$ denotes the conjugate of the quaternion, which is defined in Equation 15.

$$\mathbf{q}^* = \begin{bmatrix} \mathbf{q}_{1:3} \\ q_4 \end{bmatrix}^* \equiv \begin{bmatrix} -\mathbf{q}_{1:3} \\ q_4 \end{bmatrix} \tag{15}$$

The cross-product operator is shown in Equation 16.

$$\bar{\mathbf{q}} \otimes \mathbf{q} = \begin{bmatrix} q_4 \bar{\mathbf{q}}_{1:3} + \bar{q}_4 \mathbf{q}_{1:3} - \bar{\mathbf{q}}_{1:3} \times \mathbf{q}_{1:3} \\ \bar{q}_4 q_4 - \bar{\mathbf{q}}_{1:3} \cdot \mathbf{q}_{1:3} \end{bmatrix} \tag{16}$$

After performing the same 3D rotation on all stars, a projection of the unit vectors $[XYZ]'$ is made using the pinhole camera model (Equation 17) into the virtual sensor image plane. Stars that do not have a projection lying in this plane are eliminated from the current simulation. In Equation 17, the left vector is represented using a homography coordinate system, thus it should be normalised by w. The resulting x and y coordinates represent the projection. In Equation 17, (c_x, c_y) are the centre of the sensor plane, in pixels, and (f_x, f_y) correspond to the focal length of the lens, also in pixels.

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{17}$$

Finally, a Point Spread Function (PSF) is used to simulate the spreading of the light upon multiple pixels, and the virtual image is formed. Equation 18 shows the PSF used in Guangjun Zhang (2017). The simulator designed in this work uses an adapted version of this equation.

$$I(m,n) = \int_{m\ dx}^{(m+1)\ dx} \int_{n\ dy}^{(n+1)\ dy} \sum_{i=i}^{N} \frac{C}{2.512^{M_i}} \cdot exp\left(-\frac{(x-\bar{X}_i)^2 + (y-\bar{Y}_i)^2}{2\sigma^2}\right) dy\, dx + B \tag{18}$$

Where:

$I(m,n)$	=	Pixel value function;
(m,n)	=	Pixel coordinates (discrete);
B,C	=	Constants;
M_i	=	Magnitude of i-th star;
(x,y)	=	2D sensor frame coordinates (continuous);
$(\bar{X}_i,\bar{Y}_i)$	=	Positional mean;
σ	=	Positional standard deviation.

The adaptation of Equation 18 for the C++ language was done using the simplified form for a single star as shown in Equation 19, with the constant substitution shown in Equation 20. The omitted sum is then implemented using a loop structure, and the calculation is made for all the stars.

$$I(m,n) = \int_{m\ dx}^{(m+1)\ dx} \int_{n\ dy}^{(n+1)\ dy} D \cdot exp\left(-\frac{(x-\bar{X}_i)^2 + (y-\bar{Y}_i)^2}{2\sigma^2}\right) dy\, dx \tag{19}$$

$$D = \frac{C}{2.512^{M_i}} \tag{20}$$

While the Gaussian function for a single star can be evaluated at all (m,n) pixels, to increase the performance of the simulator, a configurable window is used to limit the neighbouring pixels considered for each star.

Equation 19 can be expressed in terms of the error function $erf(x)$ as shown in Equation 21, which was computed using Wolfram Mathematica v. 11.2.

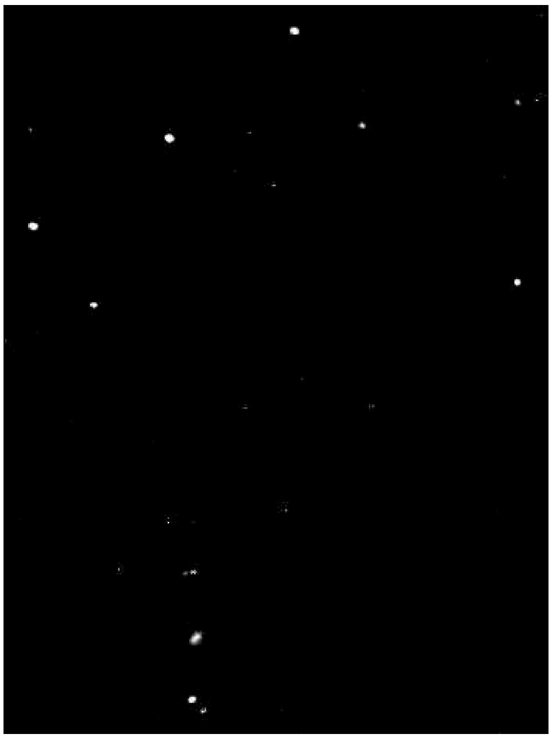
$$\begin{aligned} I(m,n) = \frac{1}{2}D\pi\sigma^2 \cdot &\left(erf\left(\frac{n-\bar{X}_i}{\sqrt{2}\sigma}\right) - erf\left(\frac{1+n-\bar{X}_i}{\sqrt{2}\sigma}\right)\right) \\ \cdot &\left(erf\left(\frac{m-\bar{Y}_i}{\sqrt{2}\sigma}\right) - erf\left(\frac{1+m-\bar{Y}_i}{\sqrt{2}\sigma}\right)\right) \end{aligned} \tag{21}$$

Equation 22 shows the mathematical definition of the error function $erf(x)$. This function is implemented in computer systems in a table format, and is present in the current C++ standard.

$$erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt \tag{22}$$

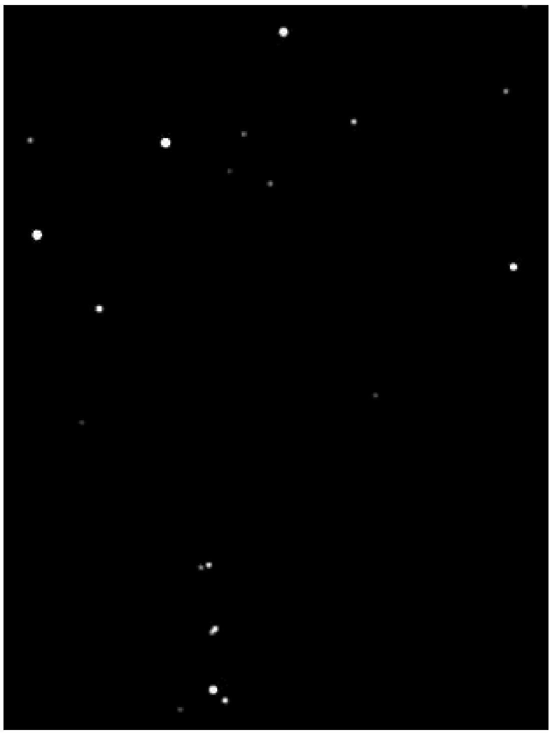
The mathematical model previously described is able to synthesise star images visually similar to images that can be captured by real hardware. A visual comparison can be done between Figure 16 and Figure 17.

Figure 16 – Real star image, captured from the ASTERIA CubeSat, JPL/NASA.



Source: Reproduced from ASTROPHYSICS... (2019).

Figure 17 – A synthetic image generated by our star simulator.



Source: Elaborated by the author.

Note: The images show part of the Orion constellation.

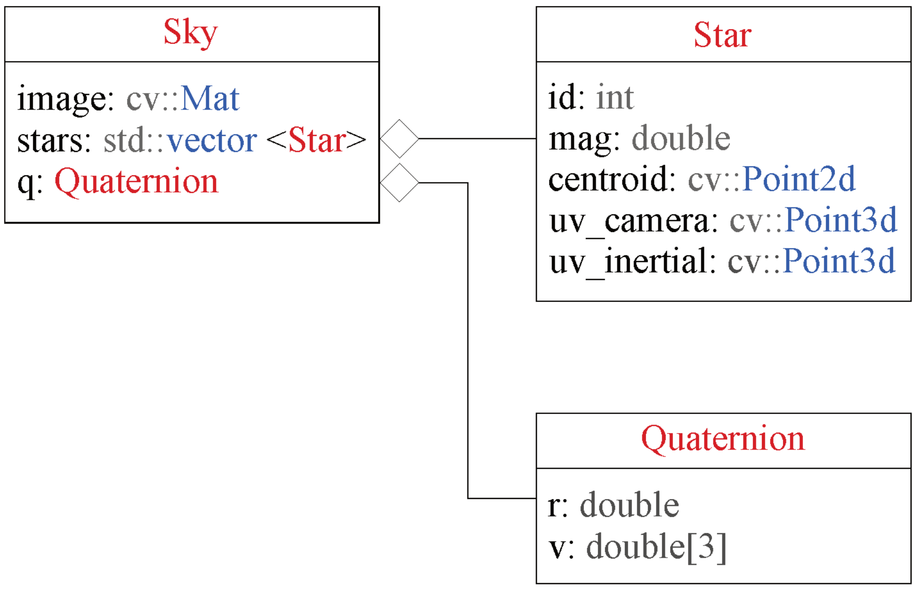
4.3.2 Universal Data Structures for Input and Output

By analysing the star tracker composing algorithms and their inputs and outputs, we constructed the data structures that are produced as the output of the star simulator. The structure can be seen in Figure 15 (a). To be considered as universal, the requirements for these structures are that they should support the verification of any combination of the composing algorithms of a star tracker, meaning centroid extraction, star identification and attitude determination. Thus, the structures should have enough information available so that it is possible for the composing algorithms to be verified as single units or associated. Another instance of the same structure can also be used to store the results of the computations performed by these algorithms, for scoring purposes.

In most cases, the types of noises used for verifying a single algorithm that works for centroid extraction, star identification or attitude determination, or an association of these algorithms, should be injected or added to specific data within the universal i/o structures, reflecting the particular test case being performed. This is discussed with more details in subsection 4.3.4.

Figure 18 shows the universal structure *Sky*. It is composed of a sky image, the list of stars present in the image and the defining attitude quaternion of its camera on the inertial frame. A *Star* is characterised by its unique identifier, its magnitude, its bi-dimensional centroid coordinate and its unit vector coordinates in both the camera and inertial frames. A *Quaternion* is defined, like its mathematical counterpart, by its real (r) and imaginary parts (i, j and k, forming *v[3]*).

Figure 18 – Universal structures for star tracker i/o interfaces.



Source: Elaborated by the author.

In short, all the algorithm blocks shown in Figure 7 will be using the same data structure as both their inputs and outputs, making the reconfiguration of test benches easier. The magnitude field can also be used for returning brightness data as long as the sign is inverted, as magnitude goes down while brightness goes up and vice versa. This ensures that posterior sort procedures on the algorithms work similarly.

4.3.3 Configurable Parameters

In order to make the verification platform configurable, to allow simulations to be made in similar testing conditions of previous researches, it was built so that the optical parameters of the virtual camera can be easily changed. Generally, three parameters can be used to define a given virtual camera. They are the field of view angle (FOV), the sensor's resolution (res) and the sensibility of the sensor, represented by the maximum visual magnitude of stars (M_{max}) that can be detected. As a convention, we use the vertical axis for calculations (y).

In the simulator, stars with higher magnitude than M_{max} are excluded from the processing. To generate a virtual camera, a camera matrix with the form shown in Equation 17 is created. The centre values (c_x, c_y) correspond to half of the resolution components ($res_x/2$, $res_y/2$).

Equation 23 defines how the focus distance, in pixels, can be calculated from the desired field of view angle (FOV_y) and the vertical resolution (res_y).

$$f = \frac{res_y}{2 \cdot \tan(FOV_y/2)} \tag{23}$$

Noise can also be added and controlled in simulations with independent configurable parameters. The types of noise that are relevant to test each component of star tracker algorithms are discussed further in subsection 4.3.4.

4.3.4 Noise Injection

For a specific test case, the injection of controlled noise should be done according to the input being expected by the algorithm. Thus, the most adequate point where noise should be added depends on what algorithm is being tested. When two adjacent algorithms are being tested (see Figure 15 (a)), for example centroid extraction and star identification, or star identification and attitude determination, the former will define the appropriated input noise, while the latter will define the scoring procedures. From the layout shown in Figure 15 (b), the outputs of each stage are kept in the universal structures. The ones that correspond to the input of the DUT can have controlled noise added to them when required.

The quality of the image acquired by the sensor is affected, in a broad perspective, by changes in the bi-dimensional location of the star projections, or changes on the brightness of stars (especially ones close to the threshold of detection of the image sensor). The amount of background noise in the image can reduce the signal to noise ratio (SNR) and bring problems such as false stars being detected by the centroid extraction algorithm. Radial and tangential distortions of the lenses, along with chromatic aberration, can also make stars appear in shifted positions on the image plane. While some of these noises are systematic, and could be minimised, for example, with a camera calibration in the laboratory, other noises can be random. Under real operating conditions the optical system could be subject to vibration or thermal variations, which would affect the lens focal length value and change the distortion pattern. Also, radiation total-dose and single-events can change the image, respectively, by raising its background noise (dark current) and by causing the emergence of hot pixels (which could mistakenly be detected as stars). Devices Under Test (DUTs) that expect an image as input will therefore need to consider these types of noises to be introduced in the image.

In the case of having star identification being evaluated in a DUT, separated from the centroid extraction step, the input of the system becomes a list of star centroids with apparent brightness information. This corresponds to the output of the (now absent) centroid extraction step. Now, the effects of the previously mentioned noises can be considered directly: absence of stars that should be detectable; presence of false stars, or even stars that are above the expected detection threshold of magnitude; errors in estimation of apparent brightness; and positional errors in the star centroid estimation. This ultimately affects the number of total identified stars and how many of those were ultimately correctly identified.

When the DUT is composed of only the static attitude determination step, its input becomes a list of stars uniquely identified. Each entry of this list is associated to the star's inertial and spacecraft centred coordinates. In this step, errors can happen due to a change in the position of the unit vectors in camera frame. They can also be caused by misidentifications, when the inertial coordinates of unrelated stars could be associated with the spacecraft coordinates of actual stars.

4.4 TESTS

A literature review of some of the existing tests performed by different authors was made, with the intuit of serving as candidates for composing the battery of tests of the platform. A summary is shown next:

1. Single star centroid estimation:
   – Error vs. noise standard deviation (KOLOMENKIN et al., 2008).
2. Star identification rate / successful attitude determination rate:
   – Percentage of correctly identified stars histogram (KOLOMENKIN et al., 2008);
   – Percentage of none/correct/ambiguous/wrong stars vs. position error (SILANI; LOVERA, 2006);
   – Percentage of none/correct/ambiguous/wrong stars vs. brightness error (SILANI; LOVERA, 2006);
   – With added false stars (KOLOMENKIN et al., 2008);
   – With 1 or 2 added false stars with brightness error (ZHAO et al., 2017);
   – With introduced position errors (PADGETT; KREUTZ-DELGADO, 1997; PADGETT; KREUTZ-DELGADO; UDOMKESMALEE, 1997; ZHAO et al., 2017; NA; ZHENG; JIA, 2009; ZHANG; WEI; JIANG, 2008);
   – With introduced brightnes noise (PADGETT; KREUTZ-DELGADO; UDOMKESMALEE, 1997; ZHAO et al., 2017; NA; ZHENG; JIA, 2009; ZHANG; WEI; JIANG, 2008);
   – With introduced focal length deviation (PADGETT; KREUTZ-DELGADO; UDOMKESMALEE, 1997);
   – Percentage of correctly identified stars vs no. of stars in the FOV (ZHANG; WEI; JIANG, 2008).
3. Bore-sight error, roll error:
   – Error vs. number of tests histogram (KOLOMENKIN et al., 2008);
   – Error vs. correctly identified stars (KOLOMENKIN et al., 2008).
4. System properties:
   – Runtime (KOLOMENKIN et al., 2008; PADGETT; KREUTZ-DELGADO, 1997; PADGETT; KREUTZ-DELGADO; UDOMKESMALEE, 1997; ZHAO et al., 2017);
   – Runtime vs. no. of stars in the FOV (ZHANG; WEI; JIANG, 2008);

– Memory requirements (KOLOMENKIN et al., 2008; PADGETT; KREUTZ-DELGADO, 1997; PADGETT; KREUTZ-DELGADO; UDOMKESMALEE, 1997; ZHAO et al., 2017).

The single star centroid estimation (1) is a test that evaluates the performance of the centroiding component of a star tracker. The star identification step is evaluated either by the (2) rate of correct identification of stars or the successful attitude determination, incorporating the attitude determination subsystem in the test. Both tests can be perturbed by the same kinds of noise, which can impact the algorithms in different ways. For example, there are algorithms which work with uncalibrated cameras (SAMAAN; MORTARI; JUNKINS, 2006), which are expected to have good performance with focal length deviation. (3) Bore-sight errors are a difference in angle between the determined attitude and the correct attitude, while the roll error is the rotation difference between them. These two tests can be applied to all parts of a star tracker. Finally, there are tests which display system properties (4), measuring the runtime of the algorithms in a given hardware and the memory consumption of the database. The memory use can be theoretically calculated by analysing the data structures involved, but in practice both tests will present different results under different hardware and operating system environment conditions, so it is difficult to present an universal comparison.

The tests can have different results with different parameters, such as the field of view (which depends on the focal length) and the sensibility of the sensor. Both will affect the number of stars visible in similar orientation conditions, which can impact the identification rates of algorithms in different ways. Therefore, both wide and restrict field of view configurations should be considered, as done in the tests in (KOLOMENKIN et al., 2008).

5 CASE STUDY DESCRIPTION AND EXPERIMENTAL RESULTS

This chapter presents practical examples that demonstrate how the verification platform can be used as an aid in the design of star trackers. From the beginning, having the sky simulator, the skeleton of the verification platform, and the inputs and outputs well defined relieves the initial work of the engineer, allowing him/her to focus on the design of the star tracker only, saving time. The benefits of the verification platform are not limited to the initial set-up though. In this section, four examples are going to be explored.

The first example, *Reproducing Existing Test Conditions*, demonstrates the ability of the verification platform to work in different test configurations.

The second example, *Computational Hot Spot Optimisations*, shows how the platform can be applied to effectively speed-up the design of star tracker through optimisations focused on the most demanding parts of the algorithms. The example focusses on reducing the runtime of the algorithms, to ultimately reduce the energy requirements.

The third and fourth examples, *Launch Environment Tests and Focal Length Noise* and *Space Environment Tests and Total Ionizing Dose*, explore how measurement of noise levels in real-world settings, measured in launch/environmental tests, can be used to calibrate the noise levels of the star simulator. Test benches are constructed, which can then be used to evaluate star tracker algorithms in similar conditions through software simulations.

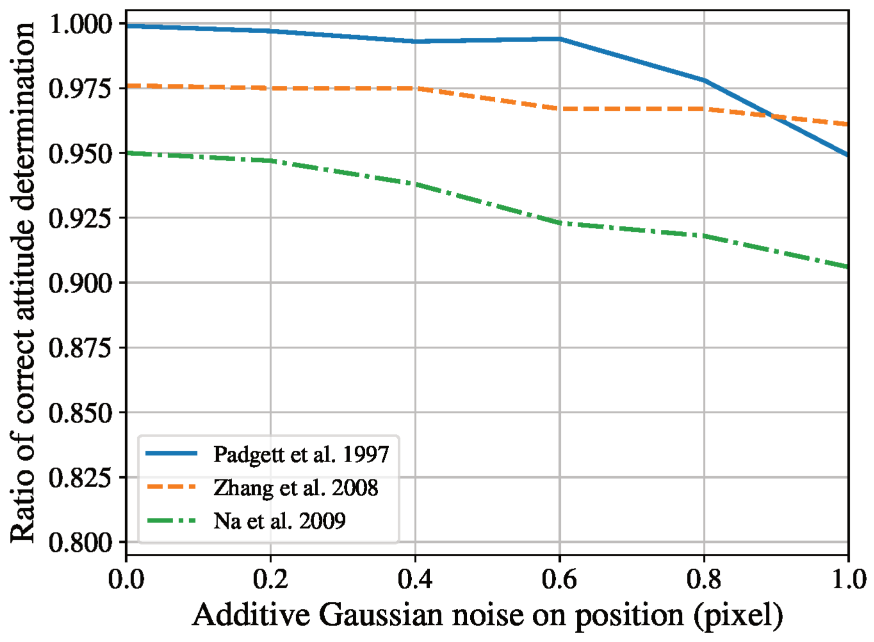
The final section, *Batch of Tests*, lists the current automated tests that were implemented on the platform, and discusses how future expansions can be made.

5.1 REPRODUCING EXISTING TEST CONDITIONS

In order to demonstrate the flexibility of the verification platform for working in different conditions, we submitted our own implementation of the Grid algorithm to similar test conditions of previous researches. The original results from these researches were shown previously in Figure 3. The reproduced results obtained with our platform can be seen in Figure 19. As some test condition parameters could not be found in the original articles, and due to differences in our implementation of the Grid algorithm and star simulator, some differences in the obtained results can be observed.

The main application of this capability is that, if working with the limitations is possible, significant time can be saved by avoiding the implementation of reference algorithms for comparisons. Instead it becomes possible to use data from other researches directly. As the platform is shared as free software, with adoption of the same tools, most differences in test conditions can be completely eliminated in the future.

Figure 19 – Behaviour of grid algorithm in different test configurations, which were reproduced from Padgett, Kreutz-Delgado, and Udomkesmalee (1997): FOV=8x8 degrees, resolution=512x512, Mv=7.5 g=40; Zhang, Wei, and Jiang (2008): FOV=12x12 degrees, resolution=1024x1024, Mv=6.0, g=40; and Na, Zheng, and Jia (2009): FOV=8x8 degrees, resolution=512x512, Mv=6.5 and g=40.



Source: Elaborated by the author.

5.2 COMPUTATIONAL HOT SPOT OPTIMISATIONS

This section shows how the verification platform can be applied to speed-up design. In short, the strategy employed is to implement a high-level software design of the system and identify regions of the program where most of the time is spent. These regions are called hot spots. When moving to lower-level implementations, the engineer only concentrate his/her attention to optimise the hot spots. This is done through redesign of the software with optimisations or through specialized hardware acceleration (e.g. FPGA). This prevents excessive work on optimising components that are not critical.

The steps followed for optimisation were:

1. High-level implementation of a reference design;
2. A test bench is created to measure the runtime of software components;
3. Identification of hot spots with proportionally high runtime in the reference design;
4. A strategy is created for optimising the hot spots;
5. Implementation of optimisations (software or hardware);

6. The same test bench for measuring runtime is applied to the optimised system, determining the effective changes.

Different test benches can also be used to ensure that the algorithm is still performing as expected for detection rates. The practical application of these steps is shown next.

5.2.1 Runtime analysis

The test bench created for the runtime tests considered a vision system with a resolution of 800×600, pixel size of 2.8 μm, and two vertical field of view (FOV) configurations: 8 and 15 degrees. FOVs configurations used for evaluating star trackers vary between authors. The values considered were selected to simultaneously try to represent popular configurations, and to allow the observation of differences in the behaviour of the algorithms when operating on narrower and wider angles. The pixel size and resolution are based on a real COTS sensor, the MT9D111(MICRON TECHNOLOGY, 2004), working at half its maximum resolution, aiming to simulate its operation. The lower resolution was chosen to facilitate debugging of hardware implementations of algorithms, due to limitations in integrated memory. The verification platform does not restrict these parameters; thus the simulation conditions can be easily changed, and the simulations redone as required.

The algorithms used on the DUT were: the *Region Growing* (ERLANK, 2013) (for centroid extraction); the *Grid Algorithm* (PADGETT; KREUTZ-DELGADO, 1997) (for star identification, with grid size $g = 24$) and *Quest* (SHUSTER; OH, 1981) (for static attitude determination). The verification platform and star simulator ran in a personal computer, and a ZedBoard development kit with a dual-core ARM Cortex A9MP (ARM v71) Zynq-7000 SoC @ 667 MHz was running the algorithms in a single thread and behaving as the DUT. The hardware-in-the-loop was implemented using a TCP/IP communication channel between the two systems. The sequence considered was a thousand random attitude configurations. For each attitude, the corresponding sensor image was synthesised by the simulator. This test case was repeated 11 times, with the first time discarded, and the runtime of each sky configuration was measured in the kit using the Chrono time library included in the C++11 standard. The repetitions were performed in order to reduce the impact of random measurement noise on samples. The operating system used was GNU/Linux, with a non-real-time kernel. Table 6 shows the average and standard deviation for the sky configurations considered.

A high standard deviation was displayed in the runtime measurements of the star identification step. This is related to the variable number of stars that can be present in a random sky image. For the *Grid Algorithm*, i.e. star identification, configurations that contain more stars will result in a longer processing time, as more catalogue look-ups need to be performed. The rate of growth of the algorithm is discussed in more detail in subsection 5.2.3.

Differently, for the centroid step, only a small difference in runtime could be observed for the different fields of view. This is an indication that the threshold operation used for seg-

Table 6 – Runtime test results (Zynq-7000 ARM Cortex A9MP (ARM v71) SoC @ 667 MHz, single thread).

		Runtime [ms]			
	FOV	*Centroid*	*Star ID*	*Attitude*	*Total*
Average	8°	13.22	1.759	0.052	15.03
Std. Dev.		0.274	1.653	0.019	1.770
Average	15°	13.78	42.71	0.082	56.57
Std. Dev.		0.352	30.50	0.018	30.75

Source: Elaborated by the author.

mentation of the stars, which considers all the pixels, predominates over the calculation of the centroids itself. Thus, the algorithm employed is more sensible to the resolution of the image then to the number of stars present in it. The standard deviation observed also supports this interpretation: even though the number of stars was changing between images, the runtime remained almost constant.

Through the analyse of the mean and the standard deviation of the runtime values, and combining with the knowledge of the structure of the algorithms used, we located two hot spots of the system: the threshold operation of the centroid step; and the catalogue lookup operation of the star identification step.

5.2.2 Improving the Centroid Extraction Step Performance

As can be seen in the results listed in Table 6, when working with a narrower FOV, the centroid extraction becomes the step with the highest consumption of resources. Thus, it is one possible target for optimisation when aiming for performance improvements of the system as a whole.

In order to achieve a better performance, a new algorithm for centroid extraction is used. Considering the observation that segmenting the star pixels from the background through threshold consumes most of the resources during centroid extraction, the strategy employed for the new algorithm's development is to apply the threshold operation for segmentation accelerated in hardware. This is done as the stream of pixels is being transmitted from the sensor. In our practical implementation, the pixels coming from the sensor through its CSI-2 interface are segmented in FPGA hardware. Subsequently, a new stream constituted of only star pixels is sent to the CPU, which then computes the star centroids.

The centroids are determined by continually filtering the incoming segmented star pixels using a first order Infinite Impulse Response (IIR) filter. The filter is described in Equation 24, where $X_n = [x_n, y_n]$ represents the input, with x_n and y_n being the coordinates of the pixels, and Y_n is the output. The gain G_n was defined by Equation 25. The optimal value of the a constant, selected to minimise the positional error, was found to be 0.8. This was determined through simulation, considering the system parameters presented in this section.

$$Y_n = G_n \cdot X_n + (1 - G_n) \cdot Y_{n-1} \tag{24}$$

$$G_n = a^n \tag{25}$$

Time is saved in development, when compared to a pure FPGA implementation, by targeting only the bottleneck of the centroiding step in hardware. The remaining operations are still performed in software using the C++ language. A pure software implementation of the new algorithm was also made with the purpose of serving as a ground truth for the comparisons. By exploring the co-verification functionality of the verification platform, it was possible to use the same *test bench* to ensure that the implemented centroid extraction algorithms were performing correctly. This was done by comparing the pure software with the hybrid version results with each other and with the reference algorithm (*Region Growing*). Small changes are expected due to the different nature of the reference algorithm and due to different numeric precision between software and hardware implementations. The values can be seen in Table 7 and Table 8. The runtime results confirm that a better execution time was achieved for the hybrid implementation. The tests were performed with 1000 random attitude configurations.

Table 7 – Comparison of centroid algorithms, with $FOV = 8°$.

	Region Growing	*Proposed (SW)*	*Proposed (SW+HW)*
Total	10014	10014	10014
Identified	9938	9923	9923
Correct	9938	9918	9918
Mean Error [px.]	0.710	0.741	0.742
Runtime [ms]	13.22	13.84	0.107

Source: Elaborated by the author.

5.2.3 Improving Star Identification Step Performance

One of the big criticisms of the *Grid Algorithm* is that, in its binary form, finding the closest match in the database requires a search which considers all the entries, resulting in $O(n)$ complexity (SPRATLING; MORTARI, 2009). The effect of this could be seen in Table 6, where increasing the FOV for the same sensitivity settings increases the number of stars in the images, quickly degrading the performance. A strategy to improve the runtime of the algorithm was employed, inspired by the *Geometric Voting Algorithm* (KOLOMENKIN et al., 2008). The angle

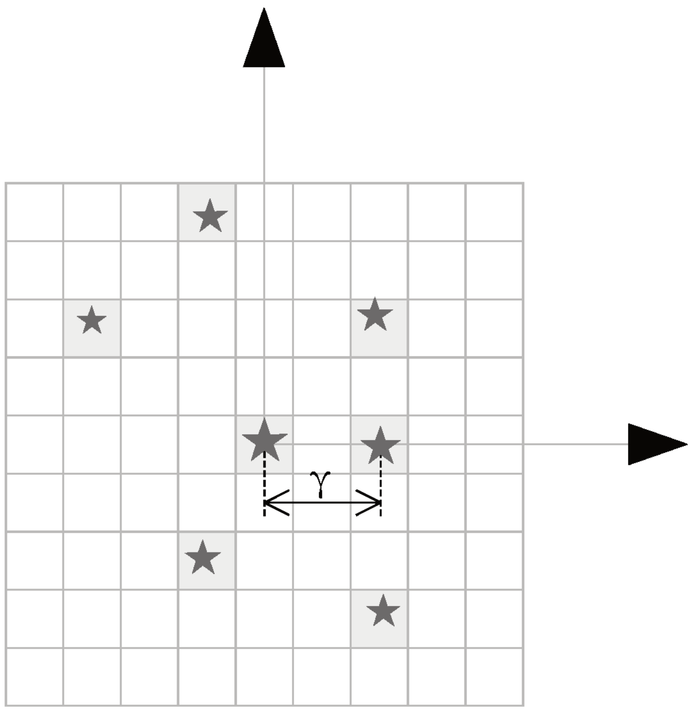
Table 8 – Comparison of centroid algorithms, with $FOV = 15°$.

	Region Growing	*Proposed (SW)*	*Proposed (SW+HW)*
Total	35006	35006	35006
Identified	34529	34332	34332
Correct	34529	34309	34309
Mean Error [px.]	0.714	0.738	0.738
Runtime [ms]	13.78	14.64	0.870

Source: Elaborated by the author.

between the star being identified and its closest neighbour (γ) is added as an additional feature (Figure 20), and the database of stars is ordered by this feature. Catalogue search algorithm complexity is then improved from $O(n)$ to $O(k)$, with k being the number of possible stars which have a neighbour with inter-star angle within the tolerance for measurement error e. The area search is effectively of stars within $[\gamma - e, \gamma + e]$. A binary descriptor is then used to further compare the reference star with the candidates within this search area in order to find the best match.

Figure 20 – Closest neighbour angular distance γ as an extra parameter.



Source: Elaborated by the author based on Ho (2012).

This technique is employed in different star identification algorithms (from the subgraph isomorphism class), for example the geometric voting algorithm (KOLOMENKIN et al., 2008).

The result is similar: by measuring the closest neighbour angle along with obtaining the star pattern, it is possible to restrict the database search within an area of $[\gamma - e, \gamma + e]$, where γ is the nearest neighbour angular distance and e is the acceptable error for the measurement.

This effectively improves the algorithm complexity from $O(n)$ to $O(k)$, with k being the number of possible stars which have a neighbour with inter-star angle within measurement tolerance, following the notation used in (SPRATLING; MORTARI, 2009).

The closest neighbour angle feature can also be used to improve the detection rates of the algorithm, in cases when there is ambiguity in the form of two or more highest-score matches with identical scores, which happens specially when there are few stars present within the pattern.

As could be expected, this modification produced a significant speed up, particularly in cases when lots of stars are present on the scene being processed. Also, for many cases, this modification increased the correct identification rate of the algorithm. This can be explained by the fact that the *Grid Algorithm* depends on the closest neighbour for achieving rotation invariance. Thus, the correct matching of the closest neighbour is a requirement for generating a correct pattern in terms of rotation. Restricting the search for patterns that have the closest neighbour star within the acceptable angular error range excludes patterns that are very unlikely to be correct, and thus increases the likelihood of correct identification. Table 9 and Table 10 show some comparison data for 8 and 15 degrees of field of view, respectively, comparing the modified algorithm with different error ranges being considered. Speed ups as high as 9.5 times were observed. Thanks to the scalability of the proposed algorithm, even higher speed ups could be achieved as the mean number of stars increases.

Table 9 – Comparison of star identification algorithms, $FOV = 8°$.

	Reference	*Binary* $e = 0.5$ *mrad*	*Binary* $e = 1.0$ *mrad*
Total	10014	10014	10014
Identified	6545 (65%)	7100 (71%)	7066 (71%)
Correct	6274 (63%)	7023 (70%)	6976 (70%)
Time μ [ms]	1.759 (1.0x)	1.014 (1.7x)	1.716 (1.0x)
Time σ [ms]	1.653	0.653	1.108

Source: Elaborated by the author.

The acceptable error e parameter should be selected with a high enough value in order to allow 2D position changes of the projections of the stars in the image sensor. Variations on position can be expected due to noise, as described previously in subsection 4.3.4. On the other hand, choosing higher values of e have a significant impact in performance, as it could be seen in Table 9 and Table 10. This happens because the database area $[\gamma - e, \gamma + e]$ that is being consider

Table 10 – Comparison of star identification algorithms, $FOV = 15°$.

	Reference	*Binary* $e = 0.5\ mrad$	*Binary* $e = 1.0\ mrad$
Total	35006	35006	35006
Identified	25624 (73%)	24557 (70%)	27280 (78%)
Correct	25099 (72%)	24032 (69%)	26892 (77%)
Time μ [ms]	42.71 (1.0x)	4.453 (9.5x)	6.965 (6.1x)
Time σ [ms]	30.50	2.414	3.641

Source: Elaborated by the author.

is larger, and the search is still being done linearly inside of it. Therefore, the sweet spot of the acceptable error *e* parameter should be high enough in order to allow the presence of positional errors, but low enough to make its introduction useful for enhancing runtime speeds.

Considering the complete software stack of the star tracker, for the cases when the addition of the closest neighbour angle (γ) feature increased the number of correctly identified stars, a higher number of correct attitude quaternions was also achieved. An important consequence for hardware requirements is that good identification rates can be realised with a smaller database. As the database size grows proportional to the square of the grid size *g*, the space freed in the database can be easily greater than the space required for the new added feature. Figure 21 demonstrates this change.

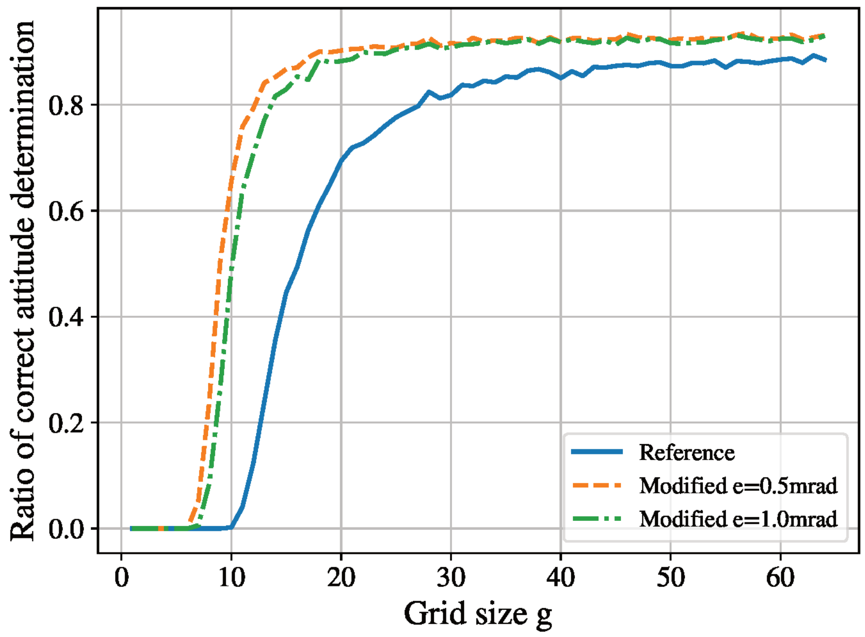
Since the modified version of the *Grid Algorithm* being evaluated could be run in less time and showed better identification rates, it is easy to jump to the conclusion that it is an improvement over the original version. However, as can be seen later in section 5.4, it requires some precautions to ensure that this stands true in real-world conditions.

5.3 IMPROVING THE SCORING FUNCTION

As discussed previously on subsection 3.2.5, the score used for the Grid algorithm does not consider false positives, false negatives or true negatives, and is based only on true positive values, as it is based on the binary AND operator. False negatives are complicated to be measured in the system, as out-of-image areas can be mistaken by non-detections, as rotation and translation operations are applied to the image, which corresponds to a confined area of the celestial sphere.

Therefore, we investigated the effects of including the false positive information for the scoring process of the binary implementation of the Grid algorithm. The number of false positives can be quickly and reliably obtained by comparing the Population Count (Popcound function, which counts the numbers of ones in a binary word) in the binary descriptor of the sensor image with the Population Count applied to the result of the AND operation between the

Figure 21 – Ratio of correct attitude quaternion determination for different grid sizes. Here, a field of view of 8° is considered.



Source: Elaborated by the author.

sensor binary descriptor and the catalogue descriptors. This can be seen in Equation 26. The second term of the operation is already known in the standard Grid algorithm. In the equation, S correspond to the sensor descriptor, D correspond to the current database descriptor being scored, and the AND operation is done bitwise.

$$False\_positives = Popcount(S) - Popcount(S\ AND\ D) \tag{26}$$

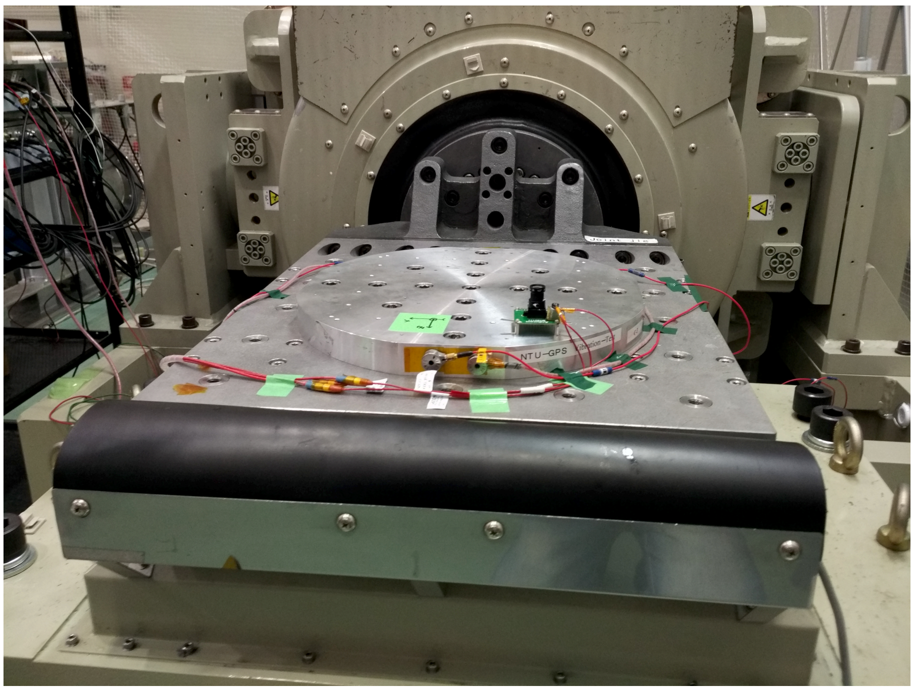
Or, in other words, if we subtract the number of common stars in both descriptors from the total number of stars detected by the sensor, the result corresponds to the false positives.

By decreasing the scoring proportionally to the number of false positives, there was an increase of 1.5% on the number of correctly identified attitude estimation of the binary implementation of the Grid algorithm. As the extra operation are done in the most critical loop of the star identification step, the impact on performance is significant. The runtime increased around 50% as a tradeoff when achieving this better attitude determination rate.

5.4 LAUNCH ENVIRONMENT TESTS AND FOCAL LENGTH NOISE

This section explains how the UVM-SystemC verification platform can be combined with data of the environment tests of the hardware. An optical system composed of the image sensor, board and lens, attached to the board using an M-12 lens holder was subject to mechanical tests to simulate the environmental conditions of a small satellite launch. The tests, which the optical system was submitted, were the quasi-static load test, random vibration test, and shock test. Before and after each test, a modal survey was also made. All of the tests follow the ISO 19683 recommended levels (ISO, 2017), and were performed in the three coordinates (*x*, *y* and *z*). The DUT can be seen undergoing the vibration tests in Figure 22. The tests were performed in the Center for Nanosatellite Testing (CeNT) laboratory of the Kyushu Institute of Technology, in Kitakyushu, Japan.

Figure 22 – Camera module undergoing vibration tests.



Source: Photo taken by the author.

Three functional tests were performed in order to ensure that the system was working properly. One was performed before the vibration tests (quasi-static load and random vibration), the second between the vibration and shock tests, and the last after the shock tests. For the procedure, a chessboard pattern was captured by the optical system from multiple angles, and the images were used to perform camera calibration of the optics using the algorithm presented in Z. Zhang (2000), through OpenCV's implementation. Thus, it was possible to measure how the focal length of the lens could change in spacecraft launch conditions. The results can be seen in Table 11.

Using the verification platform, the focal length was changed from the initial value within the range $[-2,+2]$ mm, and the percentage of correct attitude quaternions was obtained, which is shown in Figure 23.

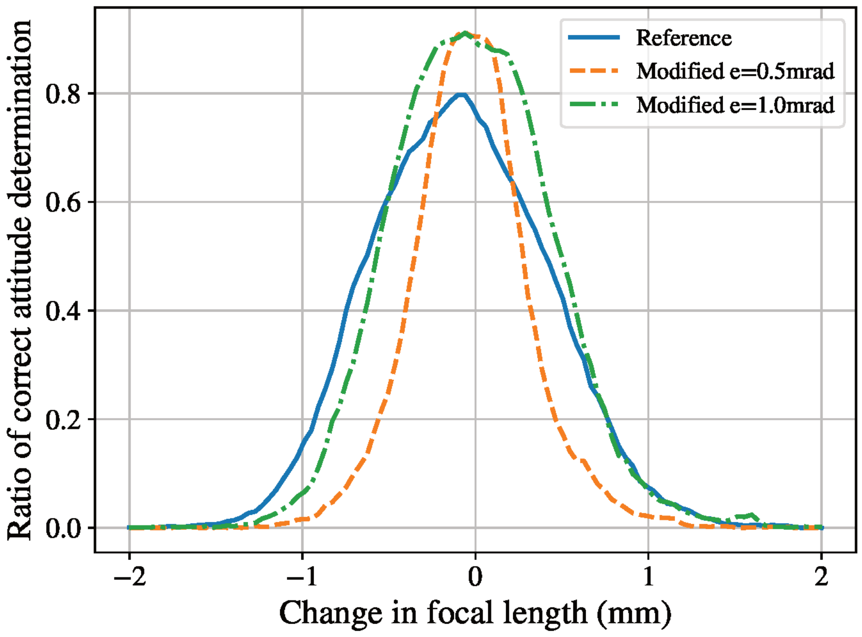
As can be seen in the Figure 23, a direct consequence of selecting a lower value of the

Table 11 – Focal length variation due to launch environment dynamics.

	f [mm]	Δ*f [mm]*
Before tests	25.287	0
After vibration	23.968	-1.3190
After shock	24.283	+0.3150

Source: Elaborated by the author.

Figure 23 – Effect of focal length deviation on the performance of reference and modified algorithms. Here, a field of view of 8° is considered.



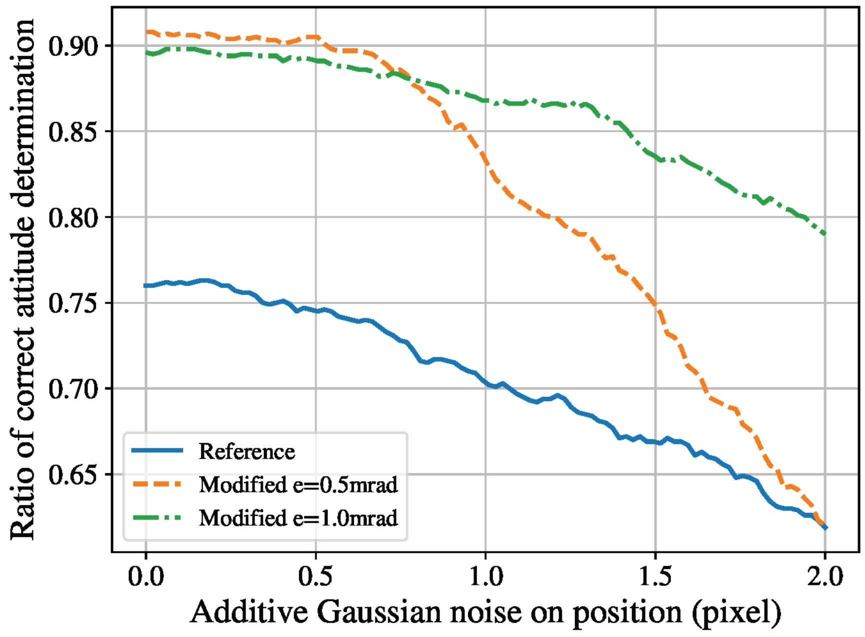
Source: Elaborated by the author.

acceptable error *e* is a smaller tolerance to 2D positional noise. This happens when the added noise changes the 2D position of the nearest neighbours are enough to make them fall outside of the area being searched in the database, thus preventing correct matches. The difference in the peak values at zero noise are a consequence of what was previously observed in Table 9. Since the number of correctly matched stars increases when the *e* parameter is used to reduce the search area of the database, the rate of correctly determined attitude quaternions also increases.

Even though the number of samples of Table 11 is not enough to predict the focal length variation of the system during launch conditions, it gives a general idea of what can be expected. By analysing Figure 23, it can be concluded that it is necessary to take precautions to improve the mechanical construction of the star tracker being designed and/or to recalibrate the system after launch. It was found that the screws present in the M-12 lens holder allow for the lens to change its focal length significantly. Also, in some cases, depending on the parameters chosen, the loss of performance of the modified star identification algorithm can be worse than the reference algorithm as the difference in focus increases. If that is the case, the mentioned precautions become even more necessary.

As other types of noises could also cause a similar behaviour when lower values of e are used, a generic 2D positional noise was also analysed. Figure 24 was generated by adding direct random 2D noise modelled by zero mean addictive Gaussian noise. It shows that, in fact, the system has increased sensibility to all noises that might change the position of the projected stars on the sensor for lower values of $e = 0.5$, but this can be mitigated when selecting a more conservative value of $e = 1.0$.

Figure 24 – Effect of noise on the 2D position of stars on the performance of reference and modified algorithms. Here, a field of view of 8° is considered.



Source: Elaborated by the author.

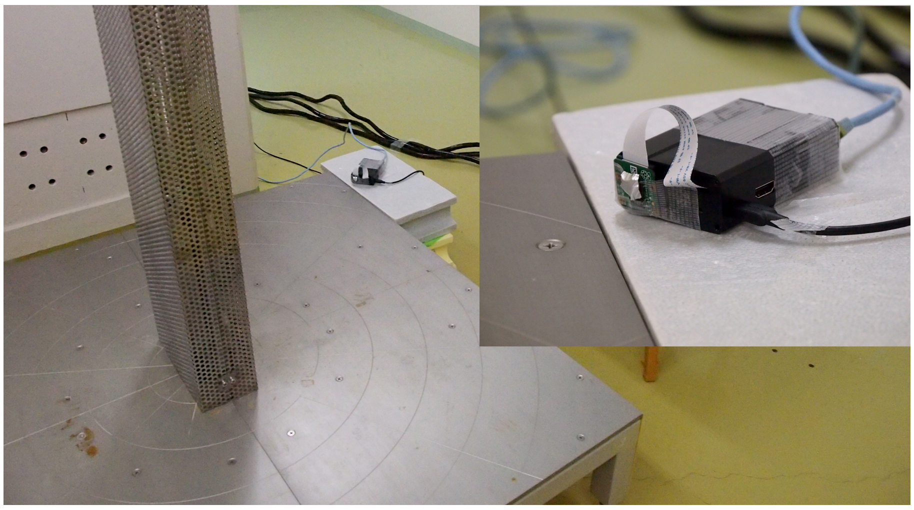
This exemplifies how the verification platform can be used to simulate the influence of specific noise on a given system. This information can then be used to review the design requirements and refine future instances of it. Different star tracker systems might have very different optical parameters, which would require specific tailored simulations for specific noise injection. With the flexibility of the black-box structure of the platform, its universal interfaces, and the ability to adapt its source code to different requirements, it is possible to reuse it to simulate a variety of different noises. These noises can be applied to different systems, with heterogeneous characteristics.

5.5 SPACE ENVIRONMENT TESTS AND TOTAL IONIZING DOSE

A Raspberry PI V2 board and a Raspberry Camera V2 were submitted to a gamma ray radiation source (Cobalt-60) in a Total Ionizing Dose (TID) test (Figure 25). The intent of the realisation of this test was to detect visual changes over the operation life of the miniaturised star tracker optical system, and subsequently reproduce these conditions with the verification platform and star simulator.

The intended total irradiation dose was 200 Gy (20 krad), which would be equivalent to 2 years operation in space in LEO conditions. The test was performed in the facilites of the Kyushu University, in Fukuoka, Japan. The permanent changes in the camera sensor were investigated by capturing dark images before and after the test. During the test, the system was operational, capturing images every 5 minutes, and sending the images through an ethernet connection to a PC.

Figure 25 – Camera module and Raspberry PI 2 board near the Cobalt-60 source, before TID test.



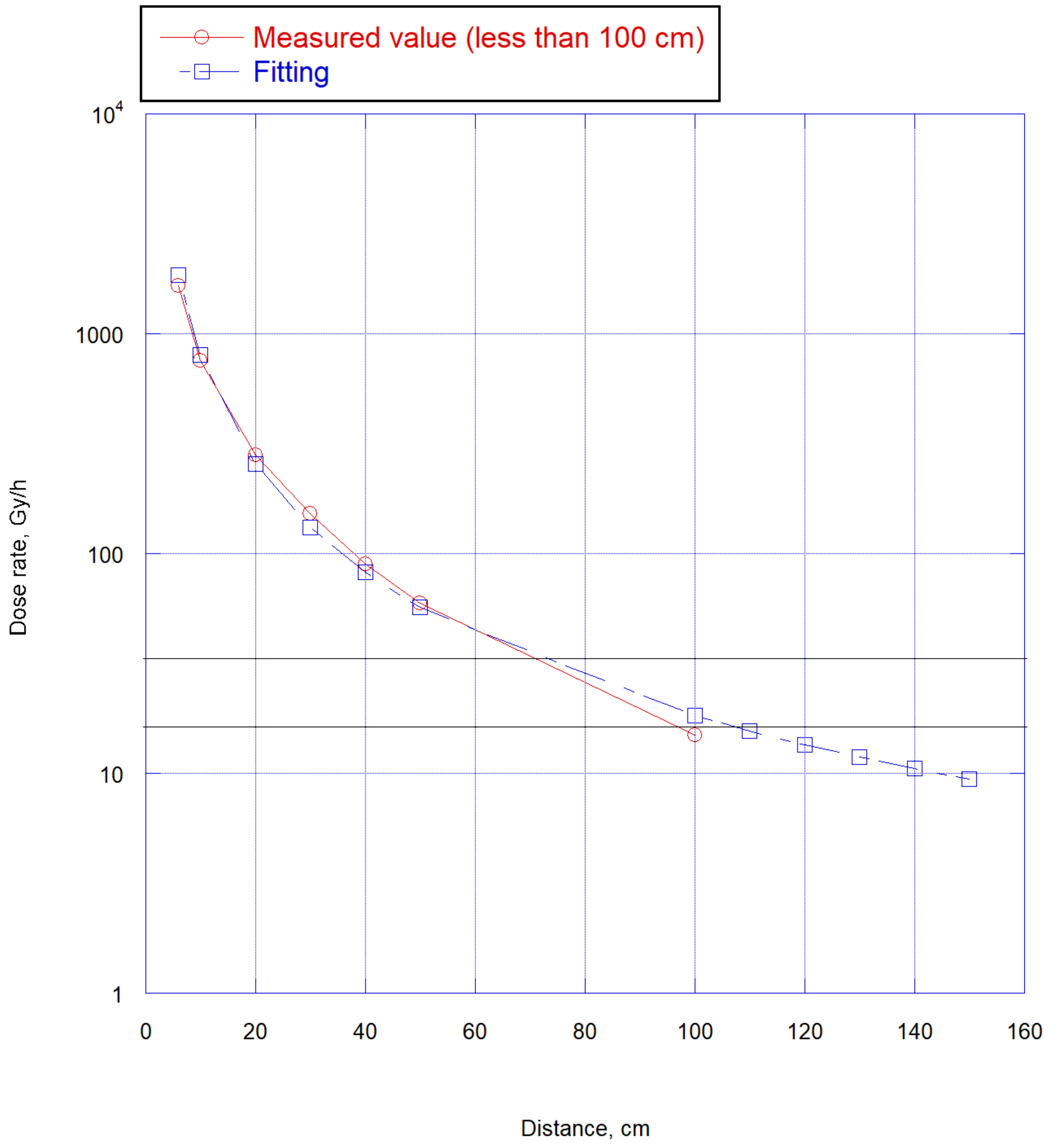
Source: Photos taken by Dr. Necmi Cihan Örger, who was conducting the test.

The DUT was separated from the radioactive source by a distance of 65cm. The received dose rate can be estimated from Figure 26 to be approximately 39 Gy/h (3.9 krad/h). The total test time was 6 hours, indicating the TID was of approximately 234 Gy (23.4 krad).

The board and camera survived the test. Post analysis of the images indicated that the mean value of pixel intensities increased slightly from 15.58 to 15.60. The pixel intensity ranges between 0-255 (8-bit). The standard deviation of pixel intensities also increased from 0.53 to 0.62. This remained consistent in subsequent tests done one week and one month after the TID, indicating that the changes were permanent.

During the TID test, the devices functioned properly. In the subsequent day of the TID test, a functional test was performed to ensure the system was operational, but the Raspberry PI

Figure 26 – Dose rate measurements over distance.



Source: Kyushu University, Fukuoka, Japan.

board did not boot successfully. After investigation, it was detected that the SD card used for the system image was permanently damaged. This did not affect the tests because the GNU/Linux system running on the Raspberry PI was operating using disk caches from RAM, and did not need to store or read any information on the SD card during operation.

By using the simulation platform with background noise simulated by Gaussian distribution tuned to the aforementioned values, it was determined that the impact of the received radiation dose on the rate of correct attitude determinations by the star tracker system is negligible. Less than 0.1% variation was detected, which is below the noise level for measurements.

Single Event Upset errors could also cause problems during the normal operation of the star tracker electronics in space applications. While performing an SEU test would also be relevant, these errors can not be simulated in the current verification platform, which focuses more on the optical system. Therefore, SEU tests were not performed in the context of this work.

5.6 BATCH OF TESTS

With the constructed verification platform and star simulator, it is possible to automate a batch of tests in order to compare different algorithms. In the current version of the platform, the following tests were automated:

1. Histogram of number of correctly identified stars.
2. Ratio of correct attitude determination under presence of false stars.
3. Ratio of correct attitude determination under focal length change.
4. Ratio of correct attitude determination under presence of star magnitude noise.
5. Ratio of correct attitude determination under presence of position noise on stars.
6. Statistics: number of correct centroid determination, star identifications and attitude determination. Error on centroid determination, star identifications and attitude determination. Runtime comparisons.

The output data is used to automatically generate graphics and tables when necessary. All of the comparison graphics shown in this chapter are examples of automated tests implemented on the platform. By expanding the number of tests implemented and considering multiple optical configurations, it would be possible to create a standard batch of tests that could be used to evaluate star tracker algorithms. The current structure of the platform can support the implementation of new tests in a quick way by exploring inheritance and polymorphism when modifying the base test declarations on the Python program that configures the tests.

6 CONCLUSIONS, REMARKS AND FUTURE WORKS

The verification platform presented in this thesis brought relevant contributions when compared to previous works: the use of a well-defined black-box structure for verification, following the Universal Verification Methodology, of which specific knowledge is transferable between different systems and scopes; the modularity of this structure, with incentives to the reusability of verification components; the ability to verify star tracker algorithms and their subcomponents separated or acting together; the ability to speed-up and assist design of software and hardware components throughout a top-down approach supporting hardware-in-the-loop configurations; and the ability to easily reproduce miscellaneous test conditions used in previous researches. These advantages can directly reduce development time and improve the range of effectiveness of the verification procedures.

The verification platform and simulator were used for straightforward tests such as determining the runtime and the number of correctly identified centroids, stars and attitude quaternions. It was also used to inject noise in the system in controllable and specialised ways, such as demonstrated in the lens focal length tests. With the aid of the platform, optimisations in software and hardware of the star tracker were achieved, demonstrating that the energy requirements of the system can be potentially reduced. Runtime for the centroiding algorithm was reduced by approximately two orders of magnitude through partial hardware acceleration in FPGA. The runtime of the star identification was also reduced around one order of magnitude by employing a different catalogue lookup strategy. In cases where higher accuracy is desirable over simple speed, it is possible to additionally implement the optimized scoring function that takes into consideration false positives during the classification.

The sharing of the star simulator and verification platform as free software is also a contribution, with the aim of being improved and reused by the scientific community. This opens up opportunities for future work in standardisation of test procedures. With a more thorough study of which test conditions and procedures could be considered as ideal to perform fair comparisons, and the current degree of automation of the platform, it would be possible to define a standard batch of tests. If the same standard procedures and common verification tools were used in researches, their results could be directly compared. As the number of algorithms increases over time, being in possession of many implementations at once makes the comparison task very difficult, hindering scientific process. On the other hand, it is simpler to generate compatible data, facilitating all future comparisons.

REFERENCES

ACCELERA SYSTEMS INITIATIVE. **Language Reference Manual**. [S.l.]: Accelera Systems Initiative, Dec. 2015. Disponível em: http://www.accellera.org/downloads/drafts-review.

ARM LIMITED. **NEON Programmer's Guide**. [S.l.: s.n.], 2013. Disponível em: http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.den0018a/index.html.

ASTROPHYSICS CubeSat Demonstrates Big Potential in a Small Package. [S.l.: s.n.]. Disponível em: https://www.jpl.nasa.gov/spaceimages/details.php?id=PIA22413. Visited on: 27 June 2019.

BARNASCONI, Martin et al. Advancing system-level verification using UVM in SystemC. In: DESIGN and Verification Conference (DVCon). [S.l.: s.n.], 2014.

BARSCHKE, Merlin et al. TechnoSat-A Nanosatellite Mission for On-Orbit Technology Demonstration, 2013.

BERLIN SPACE TECHNOLOGIES. **ST-200: Miniaturized Autonomous Star Tracker**. [S.l.]: Berlin Space Technologies, 2012. Disponível em: http://www.berlin-space-tech.com/fileadmin/media/BST_ST-200_Flyer.pdf.

BLACK, David C. et al. **SystemC: From the Ground Up, Second Edition**. 2nd. [S.l.]: Springer Publishing Company, Incorporated, 2009. ISBN 978-0-387-69957-8.

BLUE CANYON TECHNOLOGIES. **BCT Nano Star Tracker: High-Performance Attitude Determination for CubeSats**. [S.l.]: Blue Canyon Technologies, 2015. Disponível em: http://bluecanyontech.com/wp-content/uploads/2015/05/NST-Data-Sheet_1.0.pdf.

BLUE CANYON TECHNOLOGIES. **BCT TS Nano Star Tracker: High-Performance Attitude Determination for CubeSats**. [S.l.]: Blue Canyon Technologies, 2015. Disponível em: http://bluecanyontech.com/wp-content/uploads/2015/05/TS-NST-Data-Sheet_1.0.pdf.

CAMPOS, N. C. S. et al. A framework for design and validation of face detection systems. In: 2017 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON). [S.l.: s.n.], Oct. 2017. P. 1–7. DOI: 10.1109/CHILECON.2017.8229685.

DE BRUM, A. G. V. et al. The brazilian autonomous star tracker - AST. **WSEAS Transactions on Systems**, v. 12, n. 10, p. 459–470, 2013.

DZAMBA, Tom et al. Success by 1000 improvements: flight qualification of the ST-16 star tracker, 2014.

ENRIGHT, John; SINCLAIR, Doug; DZAMBA, Tom. The Things You Can't Ignore: Evolving a Sub-Arcsecond Star Tracker, 2012.

ENRIGHT, John; SINCLAIR, Doug; FERNANDO, Christy. COTS Detectors for Nanosatellite Star Trackers: A Case Study. **AIAA/USU Conference on Small Satellites**, Aug. 2011. Disponível em: https://digitalcommons.usu.edu/smallsat/2011/all2011/66.

ENRIGHT, John; SINCLAIR, Doug; GRANT, Cordell, et al. Towards star tracker only attitude estimation, 2010.

ERLANK, Alexander O.; STEYN, Willem H. Arcminute Attitude Estimation for CubeSats with a Novel Nano Star Tracker. **IFAC Proceedings Volumes**, v. 47, n. 3, p. 9679–9684, 2014.

ERLANK, Alexander Olaf. **Development of CubeStar: a CubeSat-compatible star tracker**. 2013. PhD thesis – Stellenbosch: Stellenbosch University.

ESA. The HIPPARCOS and TYCHO catalogues. Astrometric and photometric star catalogues derived from the ESA HIPPARCOS Space Astrometry Mission. In: Disponível em: http://adsabs.harvard.edu/abs/1997ESASP1200.....E. Visited on: 20 Oct. 2017.

FIALHO, Márcio Afonso Arimura; SAOTOME, Osamu. An Environment for Testing and Simulating Algorithms for Autonomous Star Sensors. In: PROCEEDINGS of COBEM. [S.l.: s.n.], 2005.

FILIPE, Nuno et al. Miniaturized Star Tracker Stimulator for Closed-Loop Testing of CubeSats. **Journal of Guidance, Control, and Dynamics**, v. 40, n. 12, p. 3239–3246, Aug. 2017. ISSN 0731-5090. DOI: 10.2514/1.G002794. Disponível em: https://arc.aiaa.org/doi/10.2514/1.G002794. Visited on: 9 Aug. 2018.

GAO, Jerry; TSAO, H.-SJ; WU, Ye. **Testing and quality assurance for component-based software**. [S.l.]: Artech House, 2003.

GAO, S.; CLARK, K., et al. Antennas for Modern Small Satellites. **IEEE Antennas and Propagation Magazine**, v. 51, n. 4, p. 40–56, Aug. 2009. ISSN 1045-9243. DOI: 10.1109/MAP.2009.5338683.

HALL, Christopher D. Spacecraft attitude dynamics and control. **Lecture Notes posted on Handouts page [online]**, v. 12, n. 2003, 2003. Disponível em: http://www.dept.aoe.vt.edu/~cdhall/courses/aoe4984/page4.html.

HARTIGAN, John A.; HARTIGAN, J. A. **Clustering algorithms**. [S.l.]: Wiley New York, 1975. v. 209.

HEIGHT, Hannibal. **A practical guide to adopting the universal verification methodology (UVM)**. [S.l.]: Lulu. com, 2013.

HO, K. A survey of algorithms for star identification with low-cost star trackers. **Acta Astronautica**, v. 73, Supplement C, p. 156–163, Apr. 2012. ISSN 0094-5765. DOI: `10.1016/j.actaastro.2011.10.017`. Disponível em: `http://www.sciencedirect.com/science/article/pii/S0094576511003195`. Visited on: 20 Oct. 2017.

HOPKINSON, G. R.; MOHAMMADZADEH, A.; HARBOE-SORENSEN, R. Radiation effects on a radiation-tolerant CMOS active pixel sensor. **IEEE Transactions on Nuclear Science**, v. 51, n. 5, p. 2753–2762, Oct. 2004. ISSN 0018-9499. DOI: `10.1109/TNS.2004.835108`.

HUA-MING, Q.; HAO, L.; HAI-YONG, W. Design and Verification of Star-Map Simulation Software Based on CCD Star Tracker. In: 2015 8th International Conference on Intelligent Computation Technology and Automation (ICICTA). [S.l.: s.n.], June 2015. P. 383–387. DOI: `10.1109/ICICTA.2015.103`.

IEEE. IEEE Standard for Standard SystemC Language Reference Manual. **IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)**, p. 1–638, Jan. 2012. DOI: `10.1109/IEEESTD.2012.6134619`.

IEEE. IEEE Standard for Universal Verification Methodology Language Reference Manual. **IEEE Std 1800.2-2017**, p. 1–472, May 2017. DOI: `10.1109/IEEESTD.2017.7932212`.

INTEL CORPORATION. **Intel® SSE4 Programming Reference**. [S.l.: s.n.], 2007. Disponível em: `https://software.intel.com/sites/default/files/m/8/b/8/D9156103.pdf`.

ISO. **ISO 19683:2017: Space systems — Design qualification and acceptance tests of small spacecraft and units**. pub-ISO: pub-ISO, 2017. Disponível em: `https://www.iso.org/standard/66008.html;%20https://webstore.ansi.org/`.

KNUTSON, Matthew W. (Matthew Walter). **Fast star tracker centroid algorithm for high performance CubeSat with air bearing validation**. 2012. Thesis – Massachusetts Institute of Technology. Disponível em: `http://dspace.mit.edu/handle/1721.1/85803`. Visited on: 12 Mar. 2018.

KOLOMENKIN, M. et al. Geometric voting algorithm for star trackers. **IEEE Transactions on Aerospace and Electronic Systems**, v. 44, n. 2, p. 441–456, Apr. 2008. ISSN 0018-9251. DOI: `10.1109/TAES.2008.4560198`.

KULU, Erik. **Nanosats Database**. en. [S.l.: s.n.], 2019. Disponível em: `https://www.nanosats.eu/index.html`. Visited on: 26 July 2019.

LEEUWEN, F. van. Validation of the new Hipparcos reduction. en. **Astronomy & Astrophysics**, v. 474, n. 2, p. 653–664, Nov. 2007. ISSN 0004-6361, 1432-0746. DOI: `10.1051/0004-6361:20078357`. Disponível em:

https://www.aanda.org/articles/aa/abs/2007/41/aa8357-07/aa8357-07.html. Visited on: 20 Oct. 2017.

LIEBE, C. C. Accuracy performance of star trackers - a tutorial. **IEEE Transactions on Aerospace and Electronic Systems**, v. 38, n. 2, p. 587–599, Apr. 2002. ISSN 0018-9251. DOI: 10.1109/TAES.2002.1008988.

LIEBE, C. C. Pattern recognition of star constellations for spacecraft applications. **IEEE Aerospace and Electronic Systems Magazine**, v. 7, n. 6, p. 34–41, June 1992. ISSN 0885-8985. DOI: 10.1109/62.145117.

LINDH, Marcus. **Development and implementation of star tracker electronics**. 2014. PhD thesis.

MARCINIAK, Martin et al. Microsatellite Star Tracker Baffles: Validation and Testing, 2013.

MARKLEY, F. Landis. Attitude Error Representations for Kalman Filtering. **Journal of Guidance, Control, and Dynamics**, v. 26, n. 2, p. 311–317, 2003. ISSN 0731-5090. DOI: 10.2514/2.5048. Disponível em: https://doi.org/10.2514/2.5048. Visited on: 20 Oct. 2017.

MARKLEY, F. Landis; CRASSIDIS, John L. **Fundamentals of spacecraft attitude determination and control**. [S.l.]: Springer, 2014. v. 33.

MCBRYDE, C. R.; LIGHTSEY, E. G. A star tracker design for CubeSats. In: 2012 IEEE Aerospace Conference. [S.l.: s.n.], Mar. 2012. P. 1–14. DOI: 10.1109/AERO.2012.6187242.

MEFENZA, Michael; YONGA, Franck; BOBDA, Christophe. Design and Verification Environment for High-Performance Video-Based Embedded Systems. In: DISTRIBUTED Embedded Smart Cameras. [S.l.]: Springer, New York, NY, 2014. P. 69–90. ISBN 978-1-4614-7704-4. DOI: 10.1007/978-1-4614-7705-1_4. Disponível em: https://link.springer.com/chapter/10.1007/978-1-4614-7705-1_4. Visited on: 7 Nov. 2017.

MEHRPARVAR, Arash et al. CubeSat Design Specification Rev 13. **The CubeSat Program, Cal Poly San Luis Obispo, US**, 2014.

MICRON TECHNOLOGY. **MT9D111 - 1/3.2-Inch System-On-A-Chip (SOC) CMOS Digital Image Sensor**. [S.l.]: Micron Technology, 2004. Disponível em: https://www.uctronics.com/download/cam_module/MT9D111_SOC2010DS.pdf.

MUJA, M.; LOWE, D. G. Fast Matching of Binary Features. In: 2012 Ninth Conference on Computer and Robot Vision. [S.l.: s.n.], May 2012. P. 404–410. DOI: 10.1109/CRV.2012.60.

NA, M.; ZHENG, D.; JIA, P. Modified Grid Algorithm for Noisy All-Sky Autonomous Star Identification. **IEEE Transactions on Aerospace and Electronic Systems**, v. 45, n. 2, p. 516–522, Apr. 2009. ISSN 0018-9251. DOI: `10.1109/TAES.2009.5089538`.

NA, Meng; JIA, Peifa. A survey of all-sky autonomous star identification algorithms. In: 2006 1st International Symposium on Systems and Control in Aerospace and Astronautics. [S.l.: s.n.], Jan. 2006. 6 pp.–901. DOI: `10.1109/ISSCAA.2006.1627471`.

OPEN SYSTEMC INITIATIVE et al. **SystemC Synthesizable Subset Draft 1.3**. [S.l.]: October, 2009. Disponível em: `http://www.accellera.org/images/downloads/drafts-review/SystemC_Synthesizable_subset.1.3.pdf`.

PADGETT, C.; KREUTZ-DELGADO, K. A grid algorithm for autonomous star identification. **IEEE Transactions on Aerospace and Electronic Systems**, v. 33, n. 1, p. 202–213, Jan. 1997. ISSN 0018-9251. DOI: `10.1109/7.570743`.

PADGETT, Curtis; KREUTZ-DELGADO, Kenneth; UDOMKESMALEE, Suraphol. Evaluation of Star Identification Techniques. **Journal of Guidance, Control, and Dynamics**, v. 20, n. 2, p. 259–267, 1997. ISSN 0731-5090. DOI: `10.2514/2.4061`. Disponível em: `https://doi.org/10.2514/2.4061`. Visited on: 20 Oct. 2017.

PALO, Scott; STAFFORD, George; HOSKINS, Alan. An agile multi-use nano star camera for constellation applications, 2013.

QUINE, Brendan M. et al. Determining star-image location: A new sub-pixel interpolation technique to process image centroids. **Computer Physics Communications**, v. 177, n. 9, p. 700–706, Nov. 2007. ISSN 0010-4655. DOI: `10.1016/j.cpc.2007.06.007`. Disponível em: `http://www.sciencedirect.com/science/article/pii/S0010465507003050`. Visited on: 20 Oct. 2017.

RADPOUR, Mohammad; SAYEDI, Sayed Masoud. SystemC-AMS modeling of photodiode based on PWL technique to be used in energy harvesting CMOS image sensor. **Integration, the VLSI Journal**, v. 60, p. 48–55, Jan. 2018. ISSN 0167-9260. DOI: `10.1016/j.vlsi.2017.08.006`. Disponível em: `http://www.sciencedirect.com/science/article/pii/S0167926017305217`. Visited on: 20 Mar. 2018.

RATH, A. W.; ESEN, V.; ECKER, W. A transaction-oriented UVM-based library for verification of analog behavior. In: 2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC). [S.l.: s.n.], Jan. 2014. P. 806–811. DOI: `10.1109/ASPDAC.2014.6742989`.

RAWASHDEH, Samir A. **Visual attitude propagation for small satellites**. 2013. PhD thesis – University of Kentucky.

SAMAAN, M.A.; STEFFES, S.R.; THEIL, S. Star tracker real-time hardware in the loop testing using optical star simulator. In: p. 2233–2245.

SAMAAN, Malak A.; MORTARI, Daniele; JUNKINS, John L. Nondimensional star identification for uncalibrated star cameras. en. **The Journal of the Astronautical Sciences**, v. 54, n. 1, p. 95–111, Mar. 2006. ISSN 0021-9142. DOI: `10.1007/BF03256478`. Disponível em: `https://link.springer.com/article/10.1007/BF03256478`. Visited on: 27 Oct. 2017.

SCHMIDT, Uwe. Astro APS - The Next Generation Hi-Rel Star Tracker Based on Active Pixel Sensor Technology. In: AIAA Guidance, Navigation, and Control Conference and Exhibit. [S.l.]: American Institute of Aeronautics and Astronautics, 2005. DOI: `10.2514/6.2005-5925`. Disponível em: `https://arc.aiaa.org/doi/abs/10.2514/6.2005-5925`. Visited on: 24 Aug. 2018.

SEGERT, Tom et al. Development of the pico star tracker ST-200–design challenges and road ahead, 2011.

SENG, Bill et al. The AeroAstro Fast-Angular-Rate Miniature Star Tracker: Algorithms and Simulation Results, 2005.

SHUSTER, M. D.; OH, S. D. Three-axis attitude determination from vector observations. **Journal of Guidance, Control, and Dynamics**, v. 4, n. 1, p. 70–77, 1981. ISSN 0731-5090. DOI: `10.2514/3.19717`. Disponível em: `https://doi.org/10.2514/3.19717`. Visited on: 1 Mar. 2019.

SIEGWART, Roland; NOURBAKHSH, Illah Reza; SCARAMUZZA, Davide. **Introduction to Autonomous Mobile Robots**. [S.l.]: MIT Press, Feb. 2011. Google-Books-ID: 4of6AQAAQBAJ. ISBN 978-0-262-01535-6.

SILANI, E.; LOVERA, M. Star identification algorithms: novel approach comparison study. **IEEE Transactions on Aerospace and Electronic Systems**, v. 42, n. 4, p. 1275–1288, Oct. 2006. ISSN 0018-9251. DOI: `10.1109/TAES.2006.314572`.

SINCLAIR INTERPLANETARY. **First Generation Star Tracker (ST-16)**. [S.l.]: Sinclair Interplanetary, 2015. Disponível em: `http://www.sinclairinterplanetary.com/startrackers/star%20tracker%202015a.pdf`.

SINCLAIR INTERPLANETARY. **Second Generation Star Tracker (ST-16RT)**. [S.l.]: Sinclair Interplanetary, 2015. Disponível em: `http://www.sinclairinterplanetary.com/startrackers/star%20tracker%20RT%202015a.pdf`.

SPACE MICRO. **Miniature Integrated Star Tracker (MIST)**. [S.l.]: Space Micro, 2015. Disponível em: `http://www.spacemicro.com/assets/datasheets/guidance-and-nav/MIST.pdf`.

SPRATLING, Benjamin B.; MORTARI, Daniele. A Survey on Star Identification Algorithms. en. **Algorithms**, v. 2, n. 1, p. 93–107, Jan. 2009. DOI: `10.3390/a2010093`. Disponível em: `http://www.mdpi.com/1999-4893/2/1/93`. Visited on: 20 Oct. 2017.

TRAWNY, Nikolas; ROUMELIOTIS, Stergios I. Indirect Kalman filter for 3D attitude estimation. **University of Minnesota, Dept. of Comp. Sci. & Eng., Tech. Rep**, v. 2, p. 2005, 2005.

VIRMONTOIS, C. et al. Radiation-Induced Dose and Single Event Effects in Digital CMOS Image Sensors. **IEEE Transactions on Nuclear Science**, v. 61, n. 6, p. 3331–3340, Dec. 2014. ISSN 0018-9499. DOI: `10.1109/TNS.2014.2369436`.

WAHBA, G. A Least Squares Estimate of Satellite Attitude. **SIAM Review**, v. 7, n. 3, p. 409–409, July 1965. ISSN 0036-1445. DOI: `10.1137/1007077`. Disponível em: `http://epubs.siam.org/doi/abs/10.1137/1007077`. Visited on: 20 Oct. 2017.

WANG, H. et al. Image smearing modeling and verification for strapdown star sensor. **Chinese Journal of Aeronautics**, v. 25, n. 1, p. 115–123, 2012. DOI: `10.1016/S1000-9361(11)60369-5`.

WERTZ, James R. **Spacecraft Attitude Determination and Control**. [S.l.]: Springer Science & Business Media, 1978. v. 73.

ZHANG, Guangjun. **Star Identification: Methods, Techniques and Algorithms**. Berlin Heidelberg: Springer-Verlag, 2017. ISBN 978-3-662-53781-7. Disponível em: `https://www.springer.com/gp/book/9783662537817`. Visited on: 1 Mar. 2019.

ZHANG, Guangjun; WEI, Xinguo; JIANG, Jie. Full-sky autonomous star identification based on radial and cyclic features of star pattern. **Image and Vision Computing**, v. 26, n. 7, p. 891–897, July 2008. ISSN 0262-8856. DOI: `10.1016/j.imavis.2007.10.006`. Disponível em: `http://www.sciencedirect.com/science/article/pii/S0262885607001990`. Visited on: 27 Oct. 2017.

ZHANG, Peng et al. A Brightness-Referenced Star Identification Algorithm for APS Star Trackers. en. **Sensors**, v. 14, n. 10, p. 18498–18514, Oct. 2014. DOI: `10.3390/s141018498`. Disponível em: `http://www.mdpi.com/1424-8220/14/10/18498`. Visited on: 20 Oct. 2017.

ZHANG, Ying et al. The computer scene generation for star simulator hardware-in-the-loop simulation. In: INTERNATIONAL Symposium on Photoelectronic Detection and Imaging 2011: Advances in Imaging Detectors and Applications. [S.l.]: International Society for Optics and Photonics, Aug. 2011. 81943h. DOI: `10.1117/12.903678`. Disponível em: `https://www.spiedigitallibrary.org/conference-proceedings-of-spie/8194/81943H/The-computer-scene-generation-for-star-simulator-hardware-in-the/10.1117/12.903678.short`. Visited on: 9 Aug. 2018.

ZHANG, Z. A flexible new technique for camera calibration. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 22, n. 11, p. 1330–1334, Nov. 2000. ISSN 0162-8828. DOI: `10.1109/34.888718`.

ZHAO, Yang et al. Real-time star identification using synthetic radial pattern and its hardware implementation. **Acta Astronautica**, v. 131, Supplement C, p. 1–9, Feb. 2017. ISSN 0094-5765. DOI: `10.1016/j.actaastro.2016.11.015`. Disponível em: `http://www.sciencedirect.com/science/article/pii/S0094576516306324`. Visited on: 20 Oct. 2017.

ZHOU, F. et al. Fast star centroid extraction algorithm with sub-pixel accuracy based on FPGA. **Journal of Real-Time Image Processing**, v. 12, n. 3, p. 613–622, 2016. DOI: `10.1007/s11554-014-0408-z`.