MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„There And Back Again.

A Data's Tale. A look into uVie-IfA's data processing."

verfasst von / submitted by

Johannes Seelig, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2020 / Vienna, 2020

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 861
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Astronomie
Betreut von / Supervisor:	Univ.-Prof. Mag. Dr. Franz Kerschbaum
Mitbetreut von / Co-Supervisor:	Mag. Dr. Roland Ottensamer

Acknowledgements

I will never be able to thank my family enough for passing on to me the passion for discovery. Without that drive for knowledge, I would never be where I am today. Thank you for never wavering to lift me up when I got down, no matter the situation.

I also thank the moon of my life, Carina, for always standing right by my side through all of the late nights and early mornings, but also for reminding me every so often that there's more out there than just work.

None of this work would have been possible without the opportunities and support given to me by the team of Franz Kerschbaum.

Last but not least, I want to thank Roland Ottensamer, who I had the privilege of working with and being advised by. Thank you for the incredible amount of support and pointers you have given me, and for always taking the time to help out with whatever issues arose.

Thank you all.

"There And Back Again.

A Data's Tale. A look into uVie-IfA's data processing."

Abstract

ESA's Solar-wind-Magnetosphere-Ionosphere-Link-Explorer (SMILE) satellite is a three year mission which aims to observe the dynamic response of the Earth's magnetosphere to the solar wind and the global impact of that response. SMILE is planned to launch with four instruments: the soft X-ray imager (SXI) which observes the magnetopause, a UV imager (UVI) which observes the earths magnetic cusps, a magnetometer (MAG) that measures the strength and direction of the local magnetic field, and a Light Ion Analyser (LIA) that records the velocity distribution of solar wind ions. This thesis concentrates on the data simulations as well as consequent on-board data analysis and event-detection specifically as it pertains to the SXI. The algorithms were written in both Python and C.

The data simulator needs to run in real time, and be adaptable enough to test various scenarios, as well as be modular so as to be implementable in future missions. This means being able to easily change event types, noise, and detector properties, while still having the simulation run in less than the expected integration time (for SMILE: 4 seconds). Due to the nature of X-Ray observations, event detection is a key step in the data processing chain of SMILE. This starts in the Front-End Electronics (FEE). With the limited on-board memory, only a 5×5 pixel package of an event is passed on from the FEE to the Data Processing Unit (DPU). Both the FEE and the DPU have a specific set of rules to determine whether a signal is an actual magnetosphere event, background X-Ray, or a cosmic ray. Once an event is confirmed, it is passed on to the On-Board-Computer (OBC) archive until the next earth-download. This thesis will take an in-depth look at the SMILE mission, the structure of the created simulator DaSi, cases of benchwork data created by DaSi, as well as the rest of the data processing chain developed for SMILE by the Institute of Astronomy at the University of Vienna.

Contents

Acknowledgements iii

Abstract v

1 Introduction 1

2 SMILE Mission 5

2.1 Mission . . . 5

2.1.1 Overview . . . 6

2.1.2 Operation Modes . . . 6

2.1.3 Orbit and Field of View . . . 9

2.1.4 Satellite . . . 9

2.2 Science . . . 12

2.2.1 Heliophysics . . . 12

2.2.2 Science Goals . . . 15

2.3 Soft X-Ray Events . . . 17

2.3.1 Composition . . . 17

2.3.2 Detection . . . 18

2.4 Instruments . . . 20

2.4.1 SXI . . . 20

2.4.2 UVI . . . 22

2.4.3 Magnetometer . . . 26

2.4.4 LIA . . . 26

3 Data Simulation 29

3.1 Background . . . 29

3.1.1 Modeling Solar wind - Magnetosphere interaction . . . 30

3.1.2 A Physical Detector . . . 33

3.2 The Simulator . . . 39

3.2.1 Configuration . . . 40

3.2.2 Coding a detector . . . 43

3.2.3 Optimization . . . 58
3.2.4 .FITS Cubes . . . 68
3.3 Comparison . . . 70
3.3.1 Open University - CCD370 . . . 70
3.3.2 StarSim - uVie . . . 71
3.3.3 ESA / ESO - Pyxel . . . 71
3.4 Future additions . . . 73

4 Application **75**
4.1 SMILE Data sets . . . 75
4.2 ARIEL . . . 76

5 Data Processing **81**
5.1 Data Chain . . . 81
5.1.1 OBC . . . 82
5.1.2 DPU . . . 82
5.1.3 IASW . . . 83
5.1.4 FEE . . . 84
5.1.5 CCD . . . 87
5.1.6 CCS . . . 90
5.2 Detection Rules . . . 92
5.2.1 FEE Data Sorting Algorithm . . . 92
5.2.2 DPU Data Sorting Algorithm . . . 93
5.3 Operation . . . 93
5.4 TL;DR . . . 103

6 Conclusion **105**

A. Appendix **111**
A.1 Zusammenfassung . . . 115
A.2 List of Figures . . . 117
A.3 List of Tables . . . 119
A.4 List of Equations . . . 121
A.5 List of Abbreviations . . . 123

Chapter 1
Introduction

Where to go? Where to go? Near and far. Here they are.

Patrick Rothfuss

There and back again. This is a tale of light. And the many adventures it goes on. That is, before figures in long robes and goggles catch the light and make lots of charts and graphs trying to understand the tales that the light tells them.

In modern astronomical research, it is notoriously difficult to run experiments. We can't touch our interests[1], we can't put a galaxy in specific scenarios, under specific conditions, to see how it will react. One either needs a decently sized telescope, or even a satellite, to obtain data needed for testing any hypothesis. It can often take years of preparation for a mission to be ready for operations. But in this time, it is necessary to test every component of the project, from the software, to the hardware. The problem is, since most missions aim to obtain data using novel methods, it is difficult to test systems before deployment. To deal with this problem, countless simulations and tests are run to account for a number of possible situations that might arise.

The goal of this thesis is to create a data simulator for the Department of Astrophysics from the University of Vienna, which can be implemented for the testing of the data processing software. The simulator was created specifically for the Solar-wind-Magneto-

[1] And even if we could, most things would be either too hot or too cold for comfort.

sphere-Ionosphere-Link-Explorer (SMILE) mission, but with the intention of having it be modular and adaptive enough to be able to implement into future missions. One other mission to which it has already started being applied to, is the Atmospheric Remote-sensing Infrared Exoplanet Large-survey (ARIEL) mission. Another key element of this simulation was for it to be efficient enough to be carried out in real time during the data processing. This means that it can be run alongside data compression, within the mission's expected observation time. Up until now, data simulators have been written for each mission individually, taking up valuable time and resources before the testing of the actual software can begin.

Many standard camera simulators already exist, such as the ESA *Pyxel* simulator (Lucsanyi et al., 2018), Penn State's ACIS simulator (Townsley and Broos, 2008), the Eddington CCD simulator (Arentoft et al., 2003), and many more (see Sec. 3.3). One really only needs to google "CCD simulator" and a plethora of open-source softwares will be readily available. However, every one that I came across was either not modular enough to create specific enough cases that were needed, or were so computationally taxing, that it could take up to a couple of minutes to expect a single image. For SMILE alone, the need for creating both benchmark data as well as specific cases, and being able to generate them on command, was clear from the beginning. Especially having only limited knowledge on the magnetospheric events that are expected to be imaged by SMILE, being able to adapt and change the types of signals being observed is of notable importance. We know quite a bit about the magnetosphere and its interactions with space weather already, but how exactly does it affect us?

Our world is becoming increasingly dependent on ever-more complex technologies. With this, the potential risk of space weather impacting our lives is more critical than ever. The Earth's magnetosphere is our primary defense against the violence of space, protecting us from cosmic and solar radiation. Many missions already exist that observe the effects of this interaction: GeoTail, Cluster, Swarm, Magnetospheric-Multi-Scale (Mukai et al., 2000; Escoubet et al., 2001; Burch et al., 2016). However, these missions have only been looking at *in situ* (local) observations of microscale processes.

Figure 1.1: *Logo for SMILE mission.*

And yet, we know surprisingly little about the global impacts of the interactions of the magnetosphere and its extraterrestrial attackers. The SMILE mission was designed to observe this link on a global scale. This is the first mission of its kind, and can provide a key link to how the sun can influence the earth's plasma environment (Sembay et al., 2015). With this knowledge, we will be better able to forecast geomagnetic storms, the most hazardous space weather events in the near-Earth environment (Branduardi-Raymont et al., 2018), affecting everything from harming spacecraft electronics, creating surges in electrical power line transmissions, and impacting GPS navigation and communication.

This thesis will first go over the SMILE mission (Fig. 1.1), its instruments, and its science goals, then take an in-depth look at the the data simulator DaSi, as well as some of the possible outputs, and end with an overview of the entire data processing chain including a user manual for its operation.

Chapter 2
SMILE Mission

There is nothing like looking, if you want to find something. You certainly usually find something, if you look, but it is not always quite the something you were after.

J. R. R. Tolkien

SMILE was proposed to observe the solar wind / magnetosphere coupling via:

1. simultaneous solar wind, magnetosheath plasma, and magnetic field measurements.
2. X-Ray images of the magnetosheath and magnetic cusps
3. UV auroral images of global auroral distributions.

This mission will complement all solar, solar-wind, and in situ magnetospheric observations. Thereby resulting in an observation of the full chain of events driving space weather for the first time ever. It is a joint mission between European Space Agency (ESA) and the Chinese Academy of Sciences (CAS).

2.1 Mission

SMILE is expected to revolutionize magnetospheric physics by providing images and movies simultaneously of the magnetopause, cusps, and auroral oval using state of the art detection techniques (Branduardi-Raymont et al., 2018).

2.1.1 Overview

SMILE aims to answer three prime science questions regarding magnetospheric physics:

1. What are the fundamental modes of the dayside solar wind/magnetosphere interaction?
2. What defines the substorm cycle?
3. How do coronal mass ejection-driven storms arise and what is their relationship to substorms?

The spacecraft is planned to be launched on a VEGA-C or shared Ariane 62 rocket and will reach a highly inclined elliptical orbit. The launch is currently scheduled for November 2023 (status as of August 26, 2020). SMILE will be three-axis stabilized so that the instruments will consistently point towards the cusp/magnetosheath and aurora. The ground segment will be shared by CAS and ESA with CAS operating the spacecraft and performing science operations for the instruments. The collection of data will be performed by ground stations of CAS and ESA respectively (Branduardi-Raymont et al., 2018).

Another goal of this mission is to establish a comprehensive outreach program to further the public understanding of these previously "invisible" solar-terrestrial processes.

2.1.2 Operation Modes

The nominal science operation along orbit will be broken into five level modes which are described in Tab. 2.1 and approximate orbit zones shown in Fig. 2.1. These modes will be supplemented by basic idle, stand-by and safe modes (SMILE, 2018a).

Mode	Description
Full Science Mode:	All 4 instruments are able to operate. Spacecraft mode driven by SXI LoS (Line of Sight) pointing to the magnetopause.
Reduced Science Mode 1:	UVI is not allowed to operate due to Sun exclusion constraints.
Reduced Science Mode 2:	SXI is not allowed to operate due to Sun exclusion constraints or because altitude is lower than 50 000 km. At that altitude, a shutter closes in front of the focal plane to limit radiation doses on the SXI CCD detector.
Reduced Science Mode 3:	Only MAG and LIA operate and the spacecraft mode is power-optimised.
X-band Communication Mode:	PLM transmits in X-band. All science and telemetry data collected along one orbit to either Troll (baseline, ESA GS, Antarctica) or Sanya (support, CAS GS, China).

Table 2.1: *Nominal science Modes of SMILE in orbit after a two month commissioning phase (Branduardi-Raymont et al., 2018).*

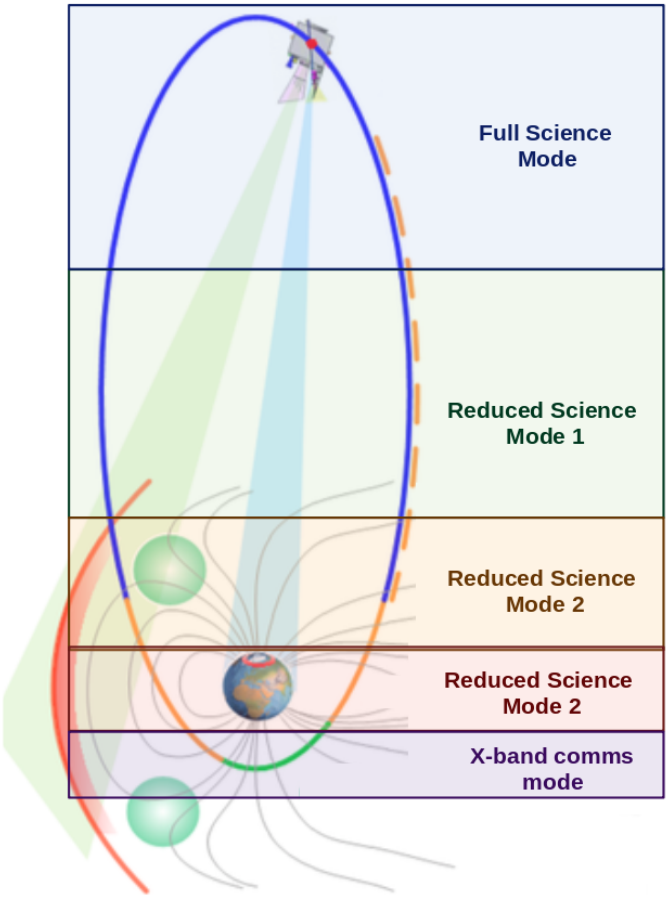

Figure 2.1: *SMILE main science operating modes. Full science mode is only possible near the highest inclination. Reduced mode 1 and 2 are without the UVI and subsequently SXI instruments. For more, see Tab. 2.1. Figure adapted from SMILE (2018a).*

Dimension	Argument
Apogee	$20\,R_E \pm 2\,R_E$
Perigee	+ 5 000 km altitude
Inclination	90 ± 27 °
Argument of Perigee	225 - 315 °

Table 2.2: *Orbit Parameters (Branduardi-Raymont et al., 2018).*

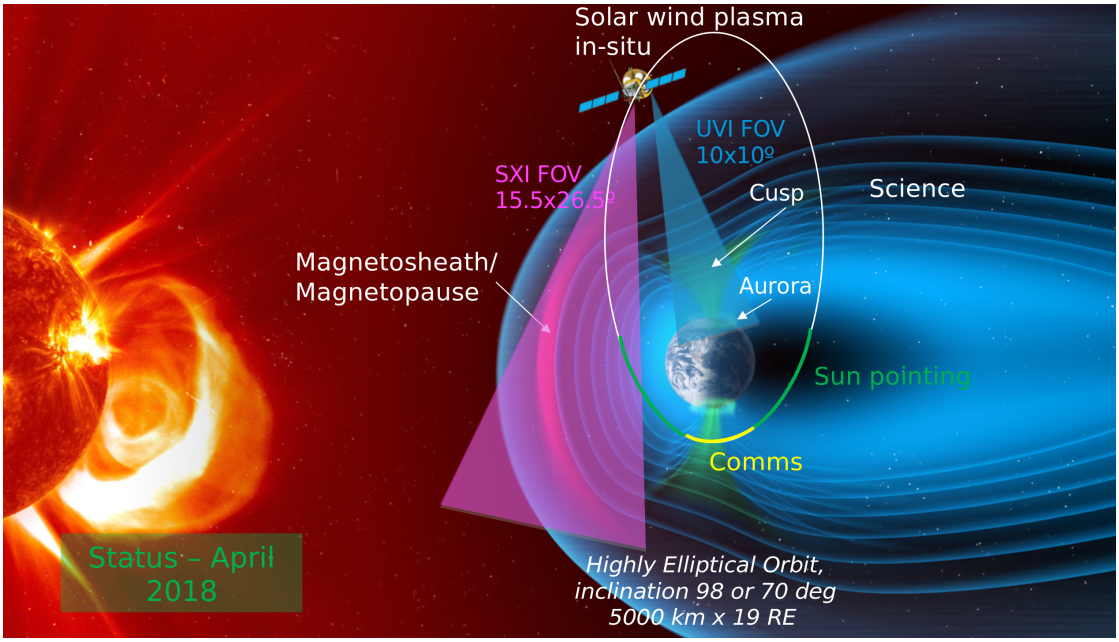

Figure 2.2: *Orbit of SMILE inclunding FoV for the SXI and UVI instruments (SMILE, 2018a).*

2.1.3 Orbit and Field of View

SMILE is set to have a highly-inclined (>70°), high apogee (20 R_E), elliptical orbit. This way, simultaneous images can be made of the solar wind impacting the magnetosheath in X-ray and auroral images made in UV. The orbit also allows for 40h continuous imaging per 51 hour orbit. The current requirements for the orbit are summarized in Tab. 2.2. The orbit including the Field of View (FoV) of the SXI and the UVI can be seen in Fig. 2.2.

2.1.4 Satellite

The SMILE satellite will be comprised of two main components. The Platform (PF), and the Payload Module (PLM). See Fig. 2.3.

Platform (PF):
The spacecraft platform encapsulates the Propulsion Module (PM) and the Service Module (SVM), see Fig. 2.4. It is under CAS supervision and the design originates from

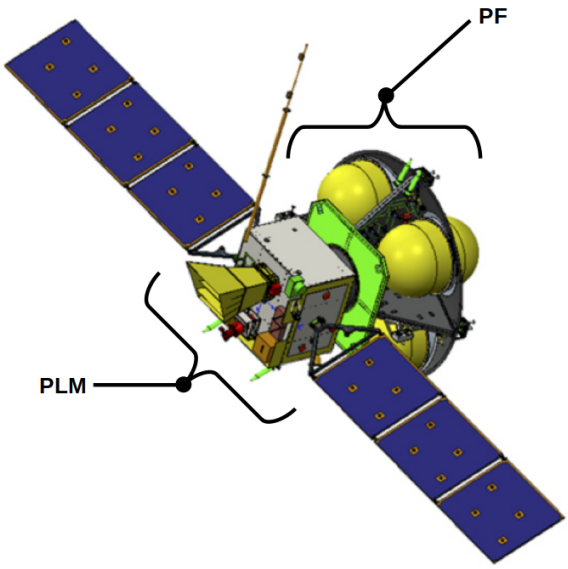

Figure 2.3: *3D rendering of the SMILE spacecraft from phase 0 study. The propulsion module (PM), the service module (SVM), as well as LIA instruments comprise the PF platform (PF). The payload module (PLM) holds the SXI, UVI and MAG (Branduardi-Raymont et al., 2018).*

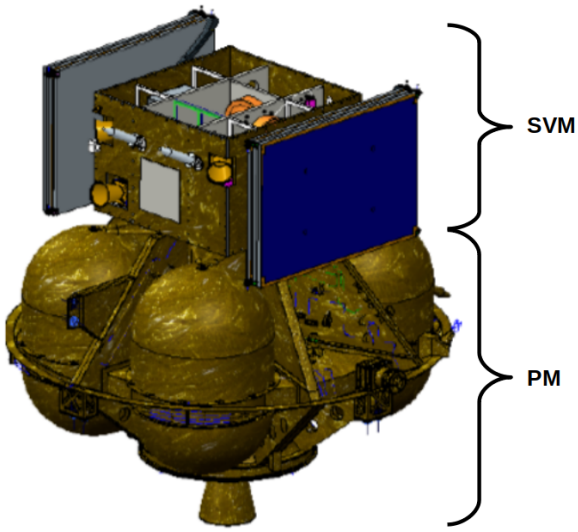

Figure 2.4: *Setup of the SMILE platform (PF) in launch configuration. The propulsion module (PM) is the bottom part and the service module (SVM), which includes the LIA instrument, is on the top (Branduardi-Raymont et al., 2018).*

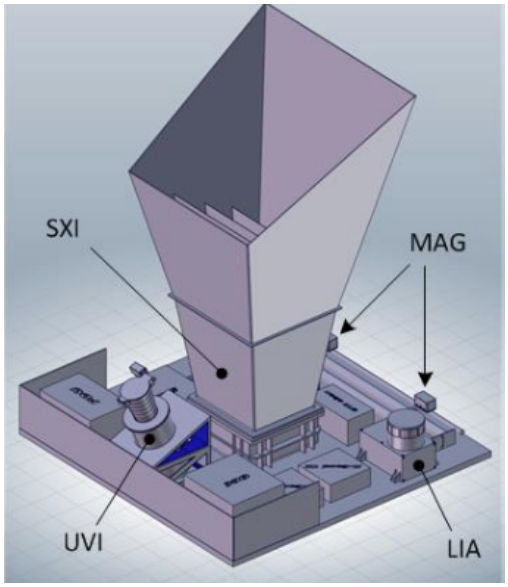

Figure 2.5: *Payload Module of SMILE hosting the SXI, UVI, and MAG instruments. Note, LIA has been moved to the SVM (Branduardi-Raymont et al., 2018).*

previous missions such as Beidu and the Dark Matter Particle Explorer. The propulsion model serves to bring SMILE to its science orbit and also hosts the LIA instruments. The service module provides power, S-Band link to the ground, and central control of the spacecraft (including PLM). The platform will have a dry mass of 547 kg (±5%) with 1 521 kg of propellant.

Payload Module (PLM):
The Payload Module (PLM) is joined at the top panel of the SVM and hosts the SXI, UVI, and MAG instruments (Fig. 2.5). The PLM's main functions are:

- Command and control of PLM units and instruments
- Interface with the Platform (e.g. telemetry, power)
- Science data storage
- Power distribution throughout PLM
- High data rate X-band transmissions to ground
- Thermal and mechanical control of PLM

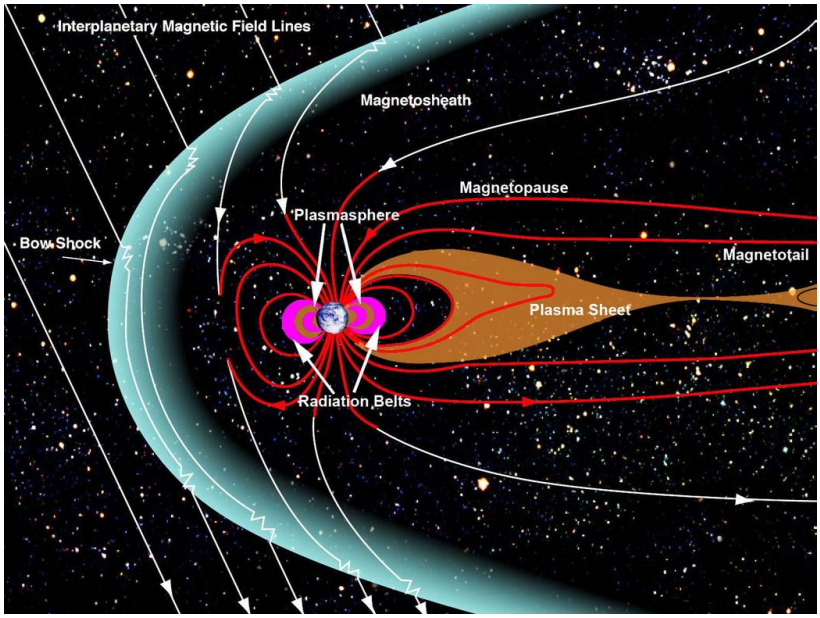

Figure 2.6: *Magnetosphere of the earth. The magnetopause (the intersection of the red and white field lines) represents the outermost boundary of the magnetosphere. The solar wind decelerates and is heated at the bow shock. The magnetosheath represents the region between the bow shock and magnetopause. The cusps are the polar regions where the solar wind enters directly into the magnetosphere. The dayside of the magnetosphere is compressed with the nightside elongated (magnetotail). From* https://www.nasa.gov/mission_pages/sunearth/science/magnetosphere2.html.

2.2 Science

2.2.1 Heliophysics

To better understand the instruments and their functions, this next part will serve as a short introduction into the physics behind Earth's magnetosphere. A summary of the most important terms can be seen in Tab. 2.3.

The *ionosphere* describes the upper atmosphere of Earth that contains a large portion of ions and (free) electrons due to ionization from from ultraviolet and x-ray radiation from the sun (from ca. 60 km up to 1 000 km). The highest part of the ionosphere forms the inner edge of the *magnetosphere*, at which point the atmosphere's charged particles are affected by the Earth's magnetic field and interact with the super-Alfvénic and supersonic

Term	Definition
Interplanetary Magnetic Field (IMF)	Component of solar magnetic field which is dragged outwards by the corona.
Magnetosphere	Region around object where charged particles are affected by the bodies magnetic field.
Solar Wind	High energy charged particles (plasma), primarily electrons and protons ejected from the Sun's corona. Typical speeds: 300 - 500 km/s, energy: 1.5 - 10 keV, pressure: 1 - 6 nPa, temperature: 1.5e6 K.
(Geo-)Magnetic storm	Disturbance of magnetosphere, induces charge exchange.
Ionosphere	Upper atmosphere containing ions and free electrons ionized from solar radiation.
Magnetopause	Boundary between magnetosphere and solar wind
Bow Shock	Boundary at which solar wind abruptly slows due to interaction with the magnetosphere of an astrophysical object.
Magnetosheath	Region between bow shock and magnetopause
Magnetotail	Dual-lobed extension of magnetosphere on night-side of celestial object.
Van Allen Belts	Two (up to three) radiation belts around Earth of solar wind particles captured in Earth's magnetic field.
Sub-Storms	Disturbance in magnetosphere. Transfer energy from solar wind to magnetosphere (and magnetotail).
Coronal Mass Ejection (CME)	Large expulsions of plasma and magnetic field from the Sun's corona.

Table 2.3: *Important terms from heliophysics. See Fig. 2.6 for visualization.*

solar wind plasma. Previous missions, such as the Japanese Geotail (Mukai et al., 2000), the European Cluster mission (Escoubet et al., 2001), the Chinese-European Double-Star project (Liu et al., 2005), the US Time History of Events and Macroscale Interactions during Substorms (THEMIS) mission (Angelopoulos et al., 2008), Van Allen probes (Mauk et al., 2013), Magnetospheric Multi-Scale mission (MMS) (Burch et al., 2016), and the Arase-Erg missions (Miyoshi et al., 2018), were geared towards precise local measurements of the various processes which progress in the magnetosphere's. These missions have increased our knowledge on the compression of the dayside magnetosphere, the drag of the nightside into a tail, reconnection of magnetic lines, and more.

The magnetosphere can be strongly deformed by a *coronal mass ejection (CME)* (Fig. 2.6). These are fast interplanetary bursts of plasma. Such CME's can induce large electrical currents inside Earth's magnetic field (also known as *geomagnetic storms*). The magnetosphere is compressed to roughly half its normal size, which in turn enhances auroral activity. Satellites (and even high flying airplanes) can be disrupted, severing communications and functionality. Such storms can even affect a city's power grid. The largest recorded CME event was on September 1, 1859 (known as the Carrington Event). A coronal ejection reached the earth in a mere 17 hours (compared to the typical three to four days) and caused widespread blackouts, disrupting telegraph communications, and resulting in auroras seen as far south as the Caribbean (Lovett, 2011).

The magnetosphere has a distinctly non-linear dynamic behavior in response to solar wind. Due to the inherent uncertainty in magnetic field line mapping, it is difficult to define the exact extent of the mangetosphere's boundaries. One method to approximate the magnetic flux is the so called Expanding-Contracting Polar Cap (ECPC) paradigm. Inferences about both the dayside magnetosphere and magnetotail can be made by measuring the structure, location, and brightness of the auroral oval (Milan et al., 2012). These have led to advancements in understanding substorms (magnetic flux which is removed from the dayside during reconnection of field lines and passed to the magnetotail lobes). Despite this theory, very few independent observations exist which confirm these processes. Observing the entire chain of events chronologically is one of the main goals of the SMILE mission.

2.2.2 Science Goals

Goal 1: Dayside reconnection

The reconnection of dayside magnetic lines has been observed to be both steady (persisting for a long time in the case of magnetic field pressure dominated conditions), as well as unsteady (bursty and time dependent due to temporal variability in the speed of the inflowing plasma or turbulence) (ESA/ATG, 2019). These reconnections cause plasma to flow anti-sunward through the magnetopause boundaries, cusps, and polar caps (Branduardi-Raymont et al., 2018). However, it is unsure how the different types of reconnection come to be, since they depend on a variety of factors such as: variations in the solar wind; cusp-topology; and even magnetotail-orientation. For example, the cusp latitude directly correlates with the level of flux inside the magnetosphere, which is driven by the reconnection process. Since this effect spans a large area of the magnetopause, it is difficult to draw quantitative conclusions from the previous in-situ observations.

Measurements required for the first science objective are (Branduardi-Raymont et al., 2018):

- steady/unsteady solar wind variations
- steady/unsteady motion of dayside magnetopause
- transient brightenings of equatorward leaps in dayside auroral oval
- transient brightness and equatorward leaps in the cusp

Goal 2: Substorm Cycle

A southward oriented Interplanetary Magnetic Field (IMF) increases the energy density of magnetotail lobes. This creates convections in the poles and magnetotail lobes (see Fig. 2.7). The exact method of energy injection, and its effect on geomagnetic activity is still debated though. One large unanswered question is, does each substorm require an individual energy injection (growth phase), or can multiple substorms result from a single growth phase (Branduardi-Raymont et al., 2018)? Also, are substorms even

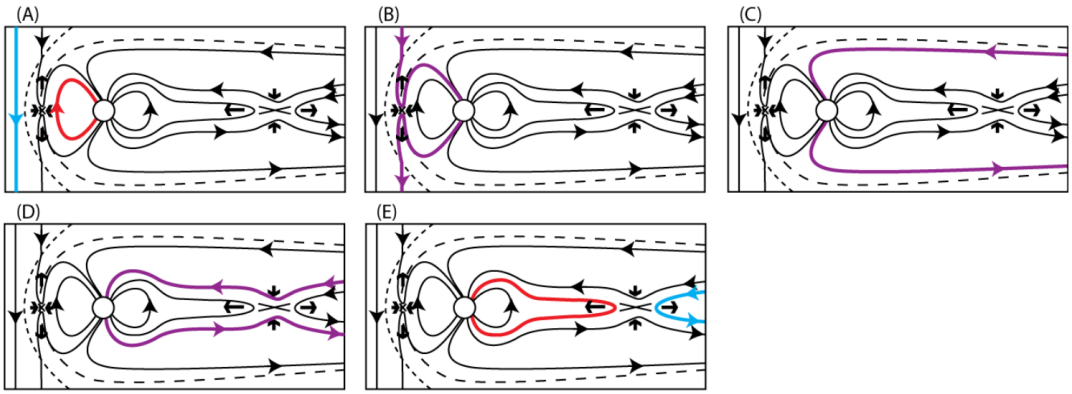

Figure 2.7: *Progression of Dungey cycle, creating substorms in magnetotail lobes. (A) Dayside reconnection, (B) magnetic flux convects over poles and (C) is stored as magnetic energy in magnetotail lobes. (D) Stored energy accumulates until explosive release. (E) Closed flux returns along with auroral effects at high altitude (Eastwood et al., 2015).*

created by changes in IMF orientation (Hsu and McPherron, 2003), or rather changes in solar wind dynamic pressure (Boudouridis et al., 2003)? Note that cases have also been observed of substorms with no obvious external driver (Huang et al., 2004).
Measurements required for the second science objective are (Branduardi-Raymont et al., 2018):

- location and motion of the dayside magnetopause boundary
- location and motion of the auroral oval
- substorm brightenings of the auroral oval
- solar wind input

Goal 3: CME driven Storms

Coronal Mass Ejections are the leading cause for large geomagnetic disturbances and are related to flow speed, field strength, and the southward component of the CME's driven magnetic field. Some known effects of CMEs are: superstorms in the equatorial ionosphere; magnetotail stretching; and joule heating in the polar ionosphere (Kozyra and et al., 2014). It is also known however, that the Earth's bow shock has the ability

to modify the magnetic field inside a CME (Turc et al., 2017). Drawing conclusions of these interactions can help define the structure of CMEs as well as further phases of geomagnetic storms.

Measurements required for the third science objective are the same as for the second science objective, only done during CME-driven storms (Branduardi-Raymont et al., 2018).

2.3 Soft X-Ray Events

Many mechanisms exist that try to explain the entry (E) and release (R) of energy and momentum within the magnetosphere. This include the Kelvin-Helmholtz instability (E), solar wind pressure variations battering the magnetosphere (E), magnetic reconnection (E+R), and plasma instabilities (R). Previous missions have been able to prove parts of these proposed mechanisms, such as reconnection, by measuring microphysical in-situ processes, but cannot distinguish global interactions of the magnetosphere and solar wind.

2.3.1 Composition

The boundaries of structures in the magnetosphere (e.g. bow shock, magnetosheath, magnetopause, and cusps) can be seen in both energetic neutral atoms (ENAs) and soft X-rays (first observed by ROSAT). The problem with ENAs is that capturing ENA images has integration times ranging from 11 to 20 hours. However, a resolution of minutes to tens of minutes is needed to capture the dynamic processes that are being looked at. For this reason, soft X-rays are chosen to image the magnetosphere (Sibeck et al., 2017). Solar wind charge exchange (SWCX) is the process of high charge state ions from the solar wind charge-exchanging with neutral ions (such as those in the Earth's geocorona). The resulting ion is in an excited state, and as it transitions back to its ground state, one or more photons are produced in either the soft X-ray or extreme ultraviolet (EUV). The relevant charge exchange reaction can be written:

$$M^{q+} + B \rightarrow M^{(q-1)+} + B^{+} \tag{2.1}$$

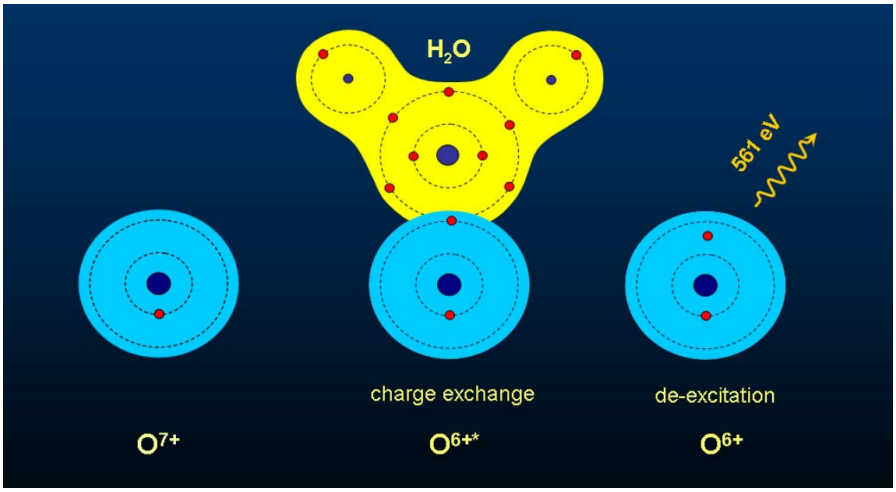

Figure 2.8: *Example of charge exchange between an O^{7+} ion with an H_2O molecule. A common cometary occurrence presented in Dennerl (2010).*

Here, the incident ion with charge q is represented as M^{q+}, with M being elements such as O, N, C, or Fe collides with a target neutral species B (Branduardi-Raymont et al., 2011). Large values for q result in the radiation of at least one photon in the EUV or soft X-ray range. An example of this would be a solar-wind ion O^{7+} colliding with a hydrogen (H) molecule (most common for Earth's exosphere), creating the reaction:

$$O^{7+} + H_2O \rightarrow O^{6+} + H_2O^{+} \tag{2.2}$$

A visual representation of O^{7+} colliding with H_2O and resulting in an X-ray line at 561 eV can be seen in Fig. 2.8.

2.3.2 Detection

Soft X-rays have an energy range of 0.05 to 2.0 keV via the charge exchange process. A high inclination, high altitude position provides an ideal vantage point for observing all relevant structures (bow shock, magnetopause, and cusps). A front-illuminated (windowless) charge-coupled device (CCD) can detect X-rays from 0.5 to 10 keV. Energies below 0.5 keV are absorbed by the chips upper gates (semiconductors for charge trans-

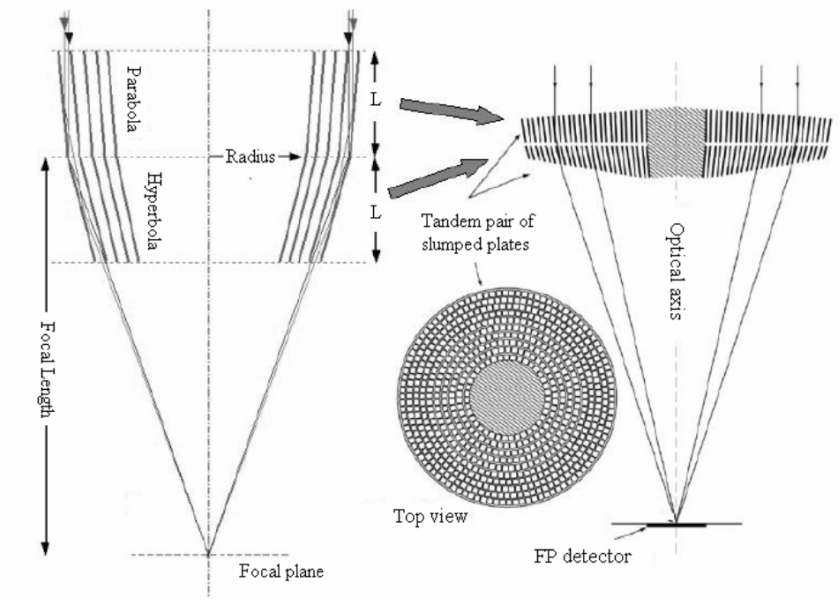

Figure 2.9: *Depiction of micro pore optics. Radially stacked pore fibres focusing the incident rays (Snowden et al., 1997).*

fer), and energies above pass through the detectores. Back-illuminated (direct) CCDs can allow for energies down to approx. 0.1 keV (Moodya et al., 2017). A thicker depletion layer could allow for capture of photons with energy higher than 10 keV. For amplifying low intensity radiation (such as X-rays), a micro-channel plate (MCP) is often incorporated into the optics. MCPs amplify particle energies by essentially translating a particle into a cloud of electrons. An MCP consists of an array of tiny slots (microchannels) that are electrically charged, when particles enter these channels, it impacts the channel wall, thereby creating a cascade of electrons which can be collected when exiting the channel (Pricea et al., 2012). To further focus the rays, micro pore optics (MPOs) can be used. These are in essence MCPs with a slight curvature of the reflecting surfaces (see Fig. 2.9).

It is important to note as well that there are other sources of soft X-rays. For example, interstellar neutral gas can enter the heliosphere, the solar system, and subsequently exchange charges with ions from the solar wind, creating similar soft X-rays. This can lead to additional X-ray emissions superimposed upon those events resulting from the

magnetosphere. Additionally, the cosmic X-ray sky emits strongly and very variably (see Fig. 2.10), sometimes even brighter than those created via SWCX (Sibeck et al., 2017). These sources must be quantified and subtracted from any soft X-ray images.

2.4 Instruments

SMILE holds four instruments which simultaneously establish solar wind properties and its effects on (and in) the magnetosphere. The Soft X-ray Imager (SXI) will observe the magnetopause, magnetosheath, and magnetospheric cusps. The Ultraviolet Imager (UVI) will image the auroral regions. The Light Ion Analyser (LIA) and the Magnetometer (MAG) will both establish solar wind properties (both local and externally).

2.4.1 SXI

The SXI (Fig. 2.12) will observe the dayside magnetospheric boundaries (bow shock, magnetopause, and cusp regions) in the soft X-ray band. This includes the location, shape, and motion of the magnetosphere, as well as being able to determine a time-dependent composition of the solar wind from the brightness of the SWCX emission lines. The instrument consists of four main units: The telescope, thermal control system, front end electronics assembly (FEE), and back end electronics assembly.

The *SXI telescope* is the image forming and detecting system, meaning the optics (MPO / straylight baffle), detector (CCD), and radiation shutter are all mounted on this unit.

The *thermal control system* is a system of thermal blankets, radiator, and radiator support structures that regulate the temperature of the SXI.

The *Front End Electronics (FEE)* are mounted at the base of the PLM and serve as communication between the detectors and back end electronics (more in Sec. 5.1.4).

The *back end electronics* includes the Data Processing Unit (DPU) (incl. boot and flight software, more in Sec. 5.1.2) and the Power Supply Unit (PSU) which provides sec-

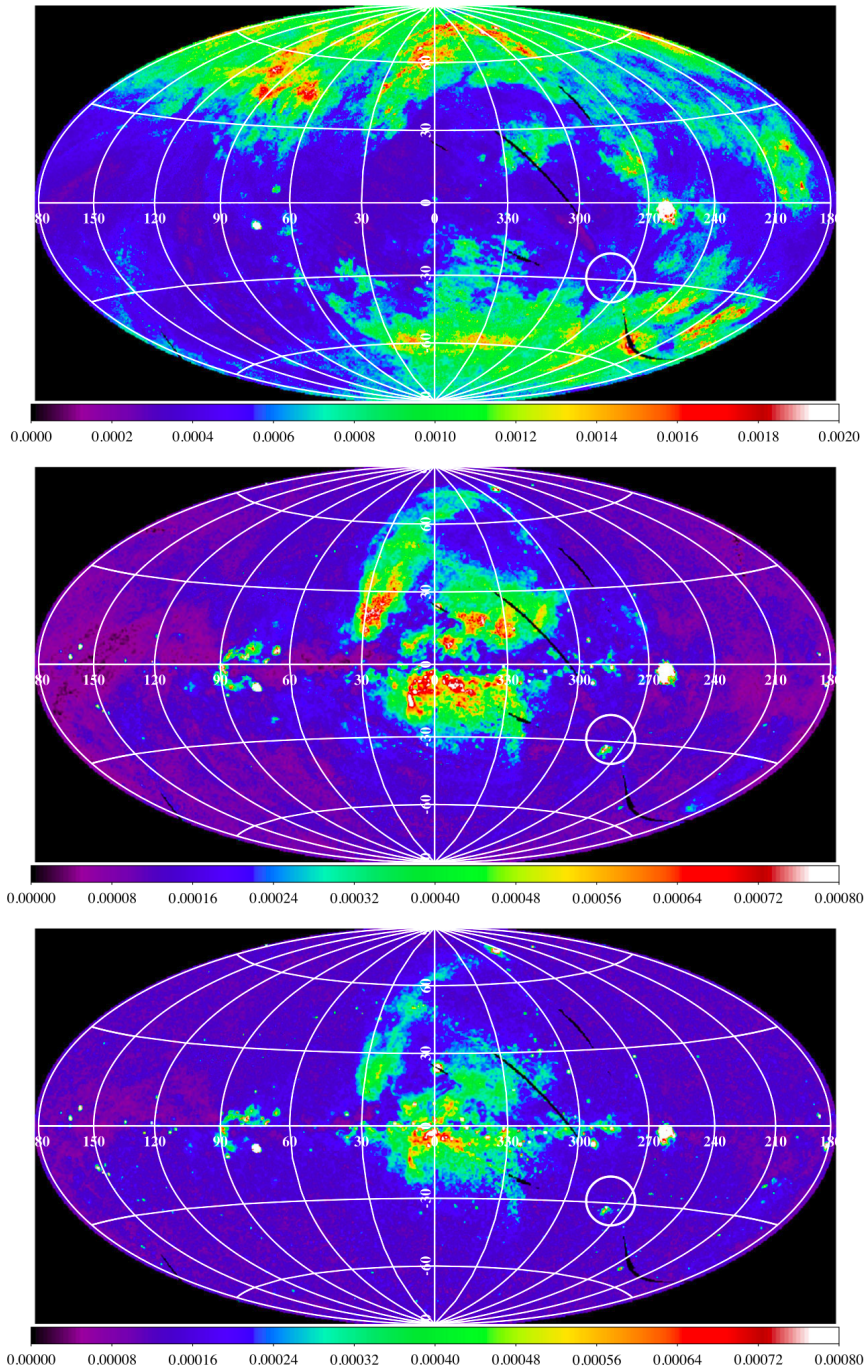

Figure 2.10: *ROSAT All-Sky Survey images of soft X-ray background as Aitoff-Hammer equal-area maps in Galactic coordinates centered on the Galactic center. Galactic longitude increases to the left, the south Galactic pole is at the bottom and the north Galactic pole is at the top. Purple and blue indicate low intensity while red and white indicate high intensity (Pricea et al., 2012). The units of the color bars are ROSAT counts s^{-1} $arcmin^{-2}$.* ***Top:*** *1/4 keV band,* ***Middle:*** *3/4 keV band,* ***Bottom:*** *1.5 keV band.*

ondary supply power to the SXI electronics. It is also mounted on the PLM base plate and interfaces with the FEE assembly, radiation shutter, PSU, and mass memory unit of the PLM.

The SXI will hold two e2vCCD370 sensors side by side (Fig. 2.11) of 4510×4510 18 μm native sized pixels (it should be noted at this point that a 719×4519 px area is reserved for asymmetric frame storage and that the detector will typically be binned by a factor of 6×6 leading to an effective pixel size of 108 μm), lobster-eye MPOs (see Sec. 2.3.2) with an additional film of aluminium to block optical and UV photons, and a straylight baffle to block sun- and earthlight from entering the aperture. A radiation shutter is also mounted which will shut during earth passes at altitudes below 50 000 km, during pointing changes, or if irregular activity is detected. Instrument parameters of the SXI can be found in Tab. 2.4. The SXI will also use 24×24 binning for the secondary science mode of observing UV photons.

The Soft X-ray Imager is developed by the University of Leicester including contributions from other UK institutes, Austria, Hungary, Spain, the Czech Republic, Norway, Ireland, and China.

2.4.2 UVI

The UVI (Fig. 2.13) will observe Earth's northern aurora, thereby allowing a connection to be made between the solar wind injection in the magnetosphere, to the activity in the cusps (Loicq et al., 2018). It is designed in such a way that it can image the aurora even in sunlit conditions. The UVI is comprised of two main hardware components:

The *UVI-C (UVI Camera Unit)* which includes the Optics Barrel Assembly (telescope optics), Detector Assembly (MCP intensifier, fiber optics, and a 1024×1024 px STAR-1000 CMOS detector), detector readout electronics (incl. interface to UVI-E), a baffle assembly, and radiator assembly.

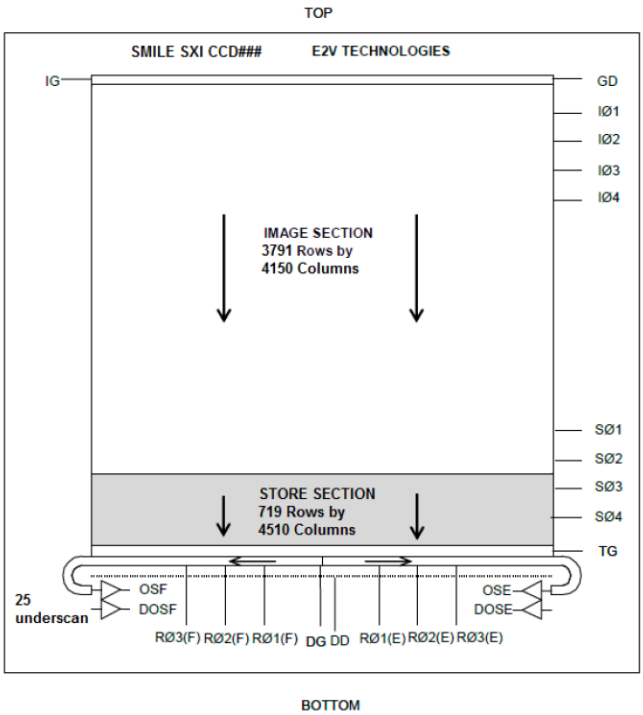

Figure 2.11: *Schematic of SMILE's SXI CCD. Procured by ESA, derivative of PLATO CCD 270 (SMILE, 2018b).*

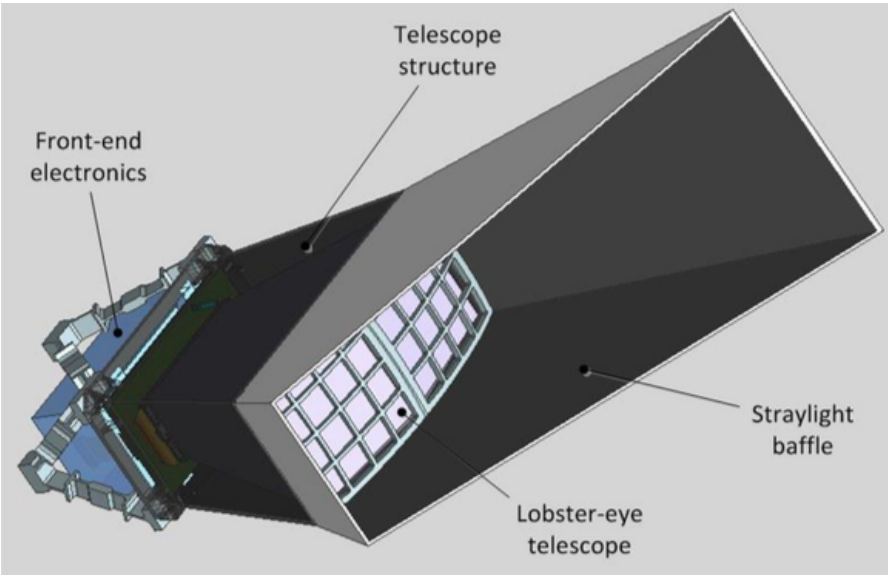

Figure 2.12: *Schematic of SMILE's SXI, developed by the University of Leicester (Branduardi-Raymont et al., 2018).*

Parameter	Value (at 0.5 keV if relevant)
Optic focal length	300 mm
MPO pore size	40 μm
MPO pore length	1.2 mm
Optic coating	Iridium
PSF FWHM	8.1 to 9.6 arcminutes
PSF HEW	~2.8 degrees
Optic total effective area	14.6 cm^2
Optic FOV	32.1° x 15.8°
Straylight baffle vignetting	~0.9
CCD QE	0.89
CCD energy resolution	50 eV (FWHM)
CCD frame integration time	~4 seconds
Filter Transmission	0.82
Total instrument effective area	9.6 cm^2
Instrument FOV	26.5° × 15.5°

Table 2.4: *Parameters of the SXI instrument (Branduardi-Raymont et al., 2018).*

The *UVI-E (Electronics Box)* which provides high-voltage power supply, thermal control electronics, and instrument control. Performance requirements for the UVI can be found in Tab. 2.5.

To account for both spatial and temporal resolution at the given altitude of 15 R_E (down to ~ 11 R_E) and a field of view (FoV) of 10 degrees and a resolution of 150 km (spatial), 60 seconds integration time was determined to be optimal.

The UVI is developed by the University of Calgary, the Canadian Space Agency, and with contributions from Belgium and China.

Field of View	10x10 degrees
Spatial Resolution	150 km @15 Re
Temporal Resolution	60 s
Spectral Band	160 - 180 nm
Detection Threshold	100 R @60 s cadence, SNR greater than 1
Dynamic Range	30 kR

Table 2.5: *UVI performance requirements (Branduardi-Raymont et al., 2018).*

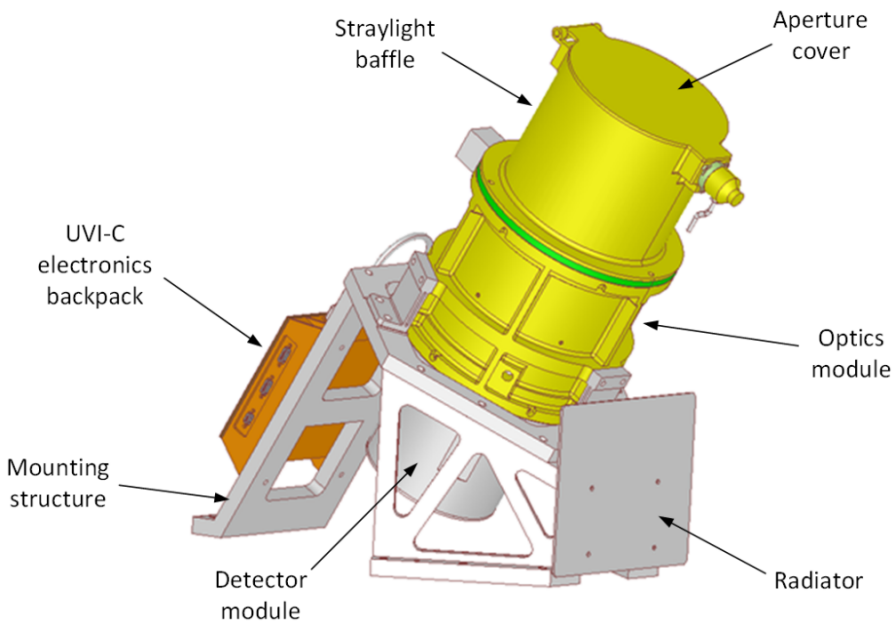

Figure 2.13: *Schematic of SMILE's UVI (Branduardi-Raymont et al., 2018).*

Dynamic range	± 12 800 nT
Digital resolution	24 bit
Sampling rate	40 Hz
Accuracy	2.0 nT
Data rate	6 kbps

Table 2.6: *MAG instrument parameters. MAG also provides a ground mode with a dynamic range of ± 64 000 nT (Branduardi-Raymont et al., 2018).*

2.4.3 Magnetometer

The magnetometer (MAG), seen in Fig. 2.14, will measure the orientation and magnitude of the local solar wind magnetic field. The MAG consists of two individual tri-axial fluxgate sensor heads on a 3 m boom (mounted on the PLM). Each fluxgate sensor head includes three ring-core fluxgate sensors mounted along three orthogonal axes to measure the three spatial components of the local magnetic field (Forslund et al., 2007). Here, a ferromagnetic core (ring) is periodically magnetically saturated via a toroidal drive, winding to modulate the local magnetic field. A solenoidal sense winding induces a current from this modulation, which can then be quantified and digitized. (Miles et al., 2019). The parameters of the MAG instrument can be found in Tab. 2.6.

The MAG is developed by the National Space Science Center in China with contributions from Austria.

2.4.4 LIA

The light ion analyzer (LIA) system, represented in Fig. 2.15, is an in-situ plasma analyzer which will measure ion distributions as well as basic moments (density, velocity, temperature tensor, and heat flux vector) of the solar wind and magnetosheath (Sul et al., 2018). The system is comprised of two identical top-hat electrostatic analyzers on opposite sides of the spacecraft platform (PF), each with a FoV of 2π. The energy analyzer (denoted as (3) in the figure), creates an electrical field between the gap of two nested electrodes (a negatively charged "inner", and a grounded "outer" hemisphere). Charged

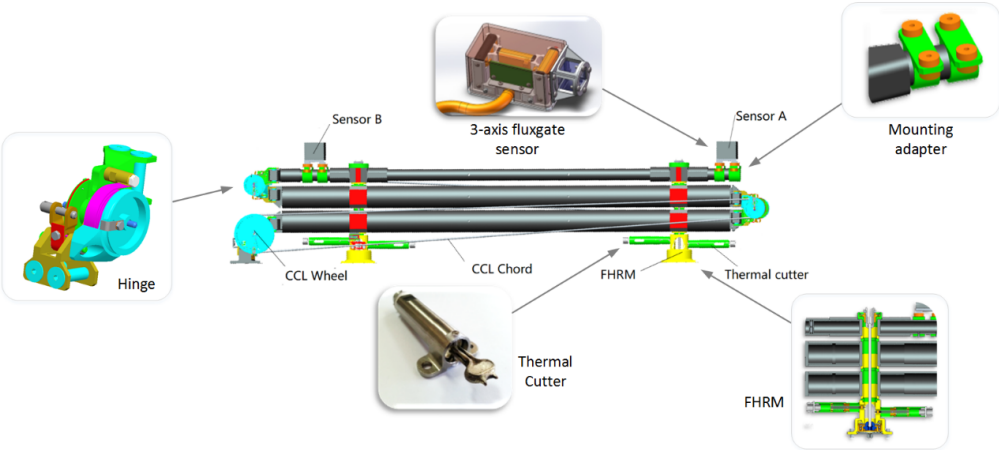

Figure 2.14: *Schematic of SMILE's Magnetometer (SMILE, 2018a).*

ions entering this field are deflected to follow the curve of the hemispheres and hit a detector. This system has an azimuthal angular resolution of down to 7.5° and can be reduced to 30°. The total analyzer has a total FoV of 360° (Azimuth) x 3° (Elevation). The main parameters of LIA can be seen in Tab. 2.7.

LIA was developed by the National Space Science Center in China with contributions from the UK and France.

Ion energy range	0.05 - 20 keV/q
Energy resolution ($\Delta E/E$)	$\leq$ 10% for 3D velocity distribution function
Field of view	2π (single instrument) 4π (full LIA system)
Time resolution	min 0.5 sec
Azimuth range (each instrument)	360°
Elevation range (each instrument)	0.05 - 13.2 keV: 0 - 90° 13.2 - 20 keV: 0 - 62°
Max. angular resolution	Azimuth: 30° coarse / 7.5° fine Elevation: $\leq$6°

Table 2.7: *LIA system parameters. Azimuthal angular resolution is a function of elevation angle (at 90° elevation, the azimuthal resolution is 26.86°) (Branduardi-Raymont et al., 2018).*

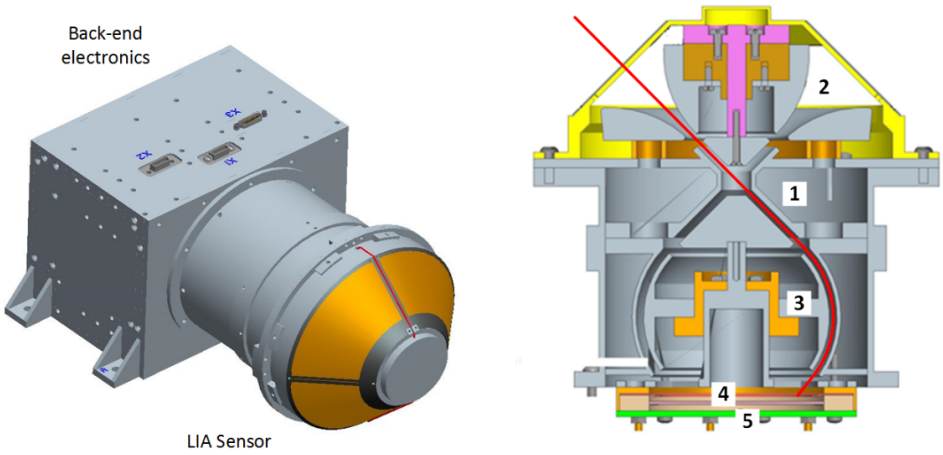

Figure 2.15: *Schematic of SMILE's LIA. Left: 3D rendition of LIA with subsystems. Right: cut-away view of electro-optical sensor. One possible path of a detected ion is shown as a red line. The energy analyzer is denoted with (3) (Branduardi-Raymont et al., 2018).*

Chapter 3
Data Simulation

Just like the simulations.

Dave Boat (SCCT)

As Ronald Fisher (1890-1962) once said: "*To consult the statistician after an experiment is finished is often merely an attempt to ask him to conduct a post mortem examination. He can, perhaps, say what the experiment died of*". One of the most common problems in astronomy is that, unlike in other sciences, nearly all objects that we want to study, are upwards of trillions of kilometers away. Another problem in the case of satellites is that, it is usually not possible to perform extensive changes after launch. This means, systems need to be rigorously tested for any possible situation that might arise.

3.1 Background

Data simulations serve as both statistical analysis tools, as well as tools for testing the robustness of a system. Simulators have the advantage to allow testing of extreme cases and catching potential bugs before deploying a mission.

SMILE's data simulator (DaSi) structure was adapted from a previous IfA simulator developed for CHaracterising Satellite (CHEOPS). It was modified to (i.) be efficient enough to act as a real-time simulator for testing the rest of SMILE's data chain (see Sec. 3.2.3), as well as (ii.) being versatile enough to be easily adapted for future missions.

Commonly, data simulations refer to models of physical or statistical processes implemented to predict observational results (see more in Sec. 3.1.1). However, since this simulator is meant to test both the hardware and software of the mission, scientific accuracy is not the primary objective. But rather to make scenarios available that the software could encounter and also test hardware limitations, such as processing speeds and storage capacities. This section will first take a quick look into already existing models of solar wind - magnetosphere interactions, the basics of a detector, and then delve into the simulation.

3.1.1 Modeling Solar wind - Magnetosphere interaction

The solar wind - magnetosphere interaction model used for SMILE was developed by the University of Science and Technology of China (USTC) and the National Space Science Center (NSSC), CAS (Hu et al., 2007). It is based on a 3-D PPMLR (extended Lagrangian piecewise parabolic global magnetohydrodynamics (MHD) model) code and solves the ideal MHD equations from -300 to 30 R_E along the x-axis and from 0 to 150 R_E in y- and z-directions (Geocentric solar magnetospheric (GSM) coordinate frame) and a grid spacing of 0.4 R_E.

X-ray intensity

The line-of-sight X-ray intensity (I) is the integration of volume emission rate (P), Eq. 3.1:

$$I = \frac{1}{4\pi}\int Pdr = \frac{1}{4\pi}\int \alpha n_H n_{\mathrm{SW}} < g > dr \;\; [keV\ cm-2\ s-1\ sr-1] \tag{3.1}$$

with α being the efficiency factor (1e-15 eV cm^2), n_H the number density of exospheric hydrogen, and n_{SW} that of the solar wind (Cravens, 2000), with integration of r from 0 to 80 R_E.

Tangential direction

The soft X-ray peak is tangent to the location of the magnetopause $\vec{p}$, which is a function of spacecraft location $\vec{s}$ and soft X-ray peak angles Θ, Eq. 3.2:

$$\vec{p} = \left| \frac{d\vec{s}}{d\Theta} \cdot \hat{\Theta} \right| \hat{r} + \vec{s} \tag{3.2}$$

with $\hat{r}$ being the unit vector along tangential line-of sight, and $\hat{\Theta}$ its respective perpendicular unit vector.

Magnetopause and Bow Shock

Boundaries for the magnetopause and bow shock, as given by the PPMLR-MHD simulations, are given by Eqs. 3.3-3.5:

$$r(\Theta, \Phi) = \frac{r_y(\Theta r_z(\Theta))}{\sqrt{[r_z(\Theta)cos(\Phi)^2] + [r_y(\Theta)sin(\Phi)^2]}} \tag{3.3}$$

$$r_y(\Theta) = r_0 \left(\frac{2}{1 + cos(\Theta)} \right)^{a_y} \tag{3.4}$$

$$r_z(\Theta) = r_0 \left(\frac{2}{1 + cos(\Theta)} \right)^{a_z} \tag{3.5}$$

Θ is the angle from the x-axis and Φ the rotation angle in right-hand direction around the x-axis starting at the y-axis. The X-ray emissivity is given by Eq. 3.6:

$$F(\vec{r}) = \begin{cases} 0 & \text{for } r < r_{\mathrm{MP}} \\ \left(A_1 + Bsin^8(\Theta)\right) \left(\frac{r}{r_{ref}}\right)^{-(\alpha + \beta sin^2(\Theta))} & \text{for } r_{\mathrm{MP}} < r < r_{BS} \\ A_2 \left(\frac{r}{r_{ref}}^{-3}\right) & \text{for } r_{\mathrm{BS}} < r \end{cases} \tag{3.6}$$

with r_{MP} and r_{BS} being the magnetopause and bow shock positions. Results of these simulations can be seen in Fig. 3.1. For more detail read Branduardi-Raymont et al. (2018).

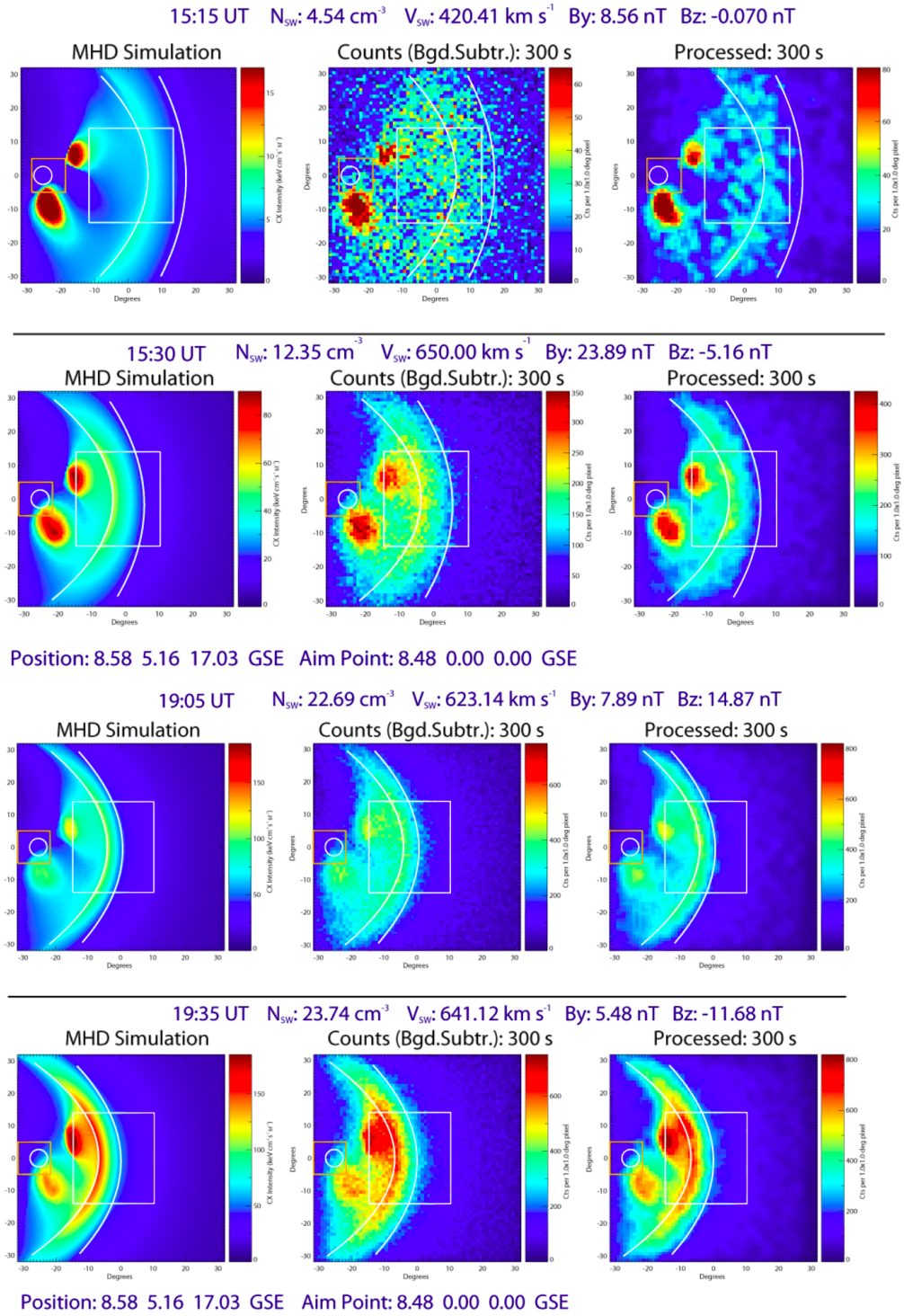

Figure 3.1: *Result of dayside solar wind - magnetosphere MHD simulation.* ***Left****: MHD simulation,* ***middle****: predicted count,* ***right****: processed image.* ***First row****: before arrive of interplanetary shock.* ***Second row****: after arrival.* ***Third/Fourth row****: response to magnetic field turning from northward (third) to southward (fourth) (Branduardi-Raymont et al., 2018). It comes as no surprise that this is where the acronym for SMILE originated.*

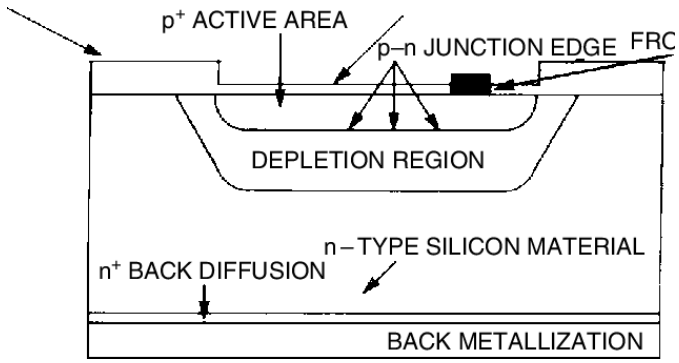

Figure 3.2: *Schematic sketch of a single front-side illuminated CCD pixel consisting of a positive "p-" doped and negative "n-" doped silicon semiconductor (Howell, 2006).*

3.1.2 A Physical Detector

As previously mentioned, and especially when considering the efficiency that this simulator requires, this in-depth modeling of the science is superfluous. Hence, more focus will from now on be put on the physical aspects of the detector itself, as well as incorporating important factors required by the processing of data. The most commonly used detectors in photometry for astrophysical missions are CCD (charge-coupled devices) and CMOSs (Complementary Metal Oxide Semiconductors). These are essentially photon-counting monochromatic detectors. This section will go over some of the most important effects that occur in such detectors, but more information can be found in (Seelig, 2018). Keep in mind that all of these characteristics need to stay dynamic so as to be adaptable for other missions.

General:

Detectors like these are arrays of pixels that store a charge proportional to the amount of photons that hit the pixel (via the photoelectric effect). Photons enter the pixel and transfer their energy to the top layer semiconductor, thereby releasing electrons directly correlated to the photon's energy. This charge is transferred to a subpixel sized electrode ("gate"), and is held in a potential well until readout. The general layout of a front-side illuminated CCD's pixel can be seen in Fig. 3.2.

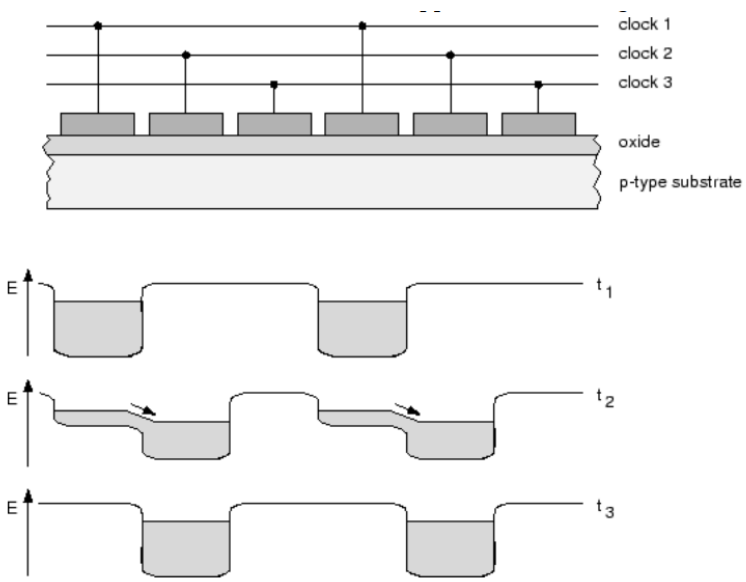

Figure 3.3: *Pixel 3-phase charge transfer. The charge is pushed along the pixels by cyclically changeing voltages at the gates (Rottinger, 1999).*

A common analogy for the readout of a CCD is that of lines of buckets along a system of conveyor belts being filled with rain water. The "buckets" (pixels) are filled with water (photons/electrons) during exposure. Then during readout, each row of buckets is emptied out into a readout-register, which is in turn emptied out into a measuring station (Fig. 3.4). Each pixel is shifted (most commonly) with a three-phase-clocking method, where each pixel consists of three gates that "push" the accumulated charge forward by cycling voltages between the gates, thereby letting the charge fall to the gate with the largest potential well, i.e. highest voltage (Fig. 3.3). Once in the readout register, the charge is shifted to an analog-to-digital converter (ADC) where it is quantified via correlated double sampling[1] (CDS). Note, in the case of a CMOS detector, A-D conversion is done on each individual pixel through attached transistors. Quantified values are referred to as ADUs (analog-to-digital units), and can be directly reverted to an amount of electrons via the gain (see below).

[1]CDS: a capacitor samples the charge differential to a reset voltage.

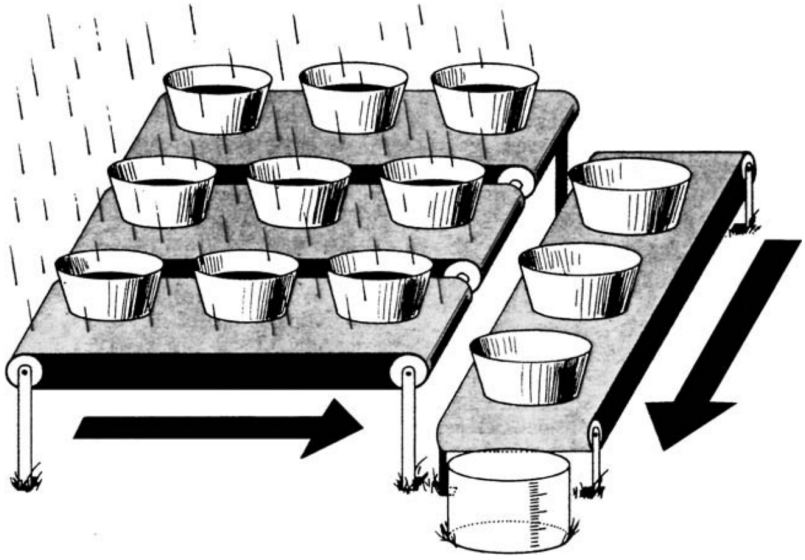

Figure 3.4: *Analogy of CCD sensor read out. The "measuring station" is the analog-to-digital converter (ADC) which quantifies the incoming charge via correlated double sampling (CDS) (Janesick, 1987).*

Characteristics

The main characteristics of a CCD can be summarized as:

- Spectral Response / Quantum Efficiency
- Bias
- Readout Noise
- Dark Current
- Gain / Dynamic Range
- Linearity

Spectral Response:
The spectral response (SR) defines the sensitivity of a detector as a function of wavelength. This sensitivity per wavelength is defined as the quantum efficiency (QE). The QE of a detector describes the ratio of photons that encounter the sensor, with the number of photons that are actually registered. An ideal sensor would therefore have a QE

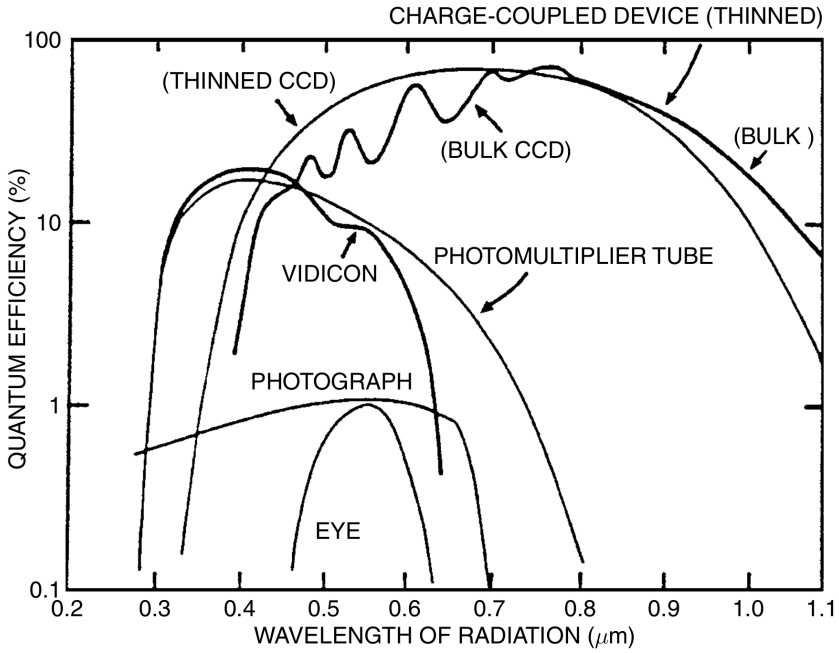

Figure 3.5: *Common spectral response of different detectors. The sensitivity at each wavelength is defined as the quantum efficiencies (Howell, 2006).*

of 100%. An eye typically has a maximum QE of 1%, and most CCDs currently have a max QE in the range of 60-80%. Common spectral responses are presented in Fig. 3.5.

Bias:

One would assume that, if a CCD were to be read out without any exposure, all pixels would yield a value of zero. This is not the case. Each pixel is given a small offset value. This is in order to avoid potential negative values[2]. The added benefit of avoiding negative numbers lies in the fact that ADCs are bit counters, and such bit counters typically assign the highest bit as a representation of sign (0 for positive or 1 for negative values respectively). If, for example, a 16-bit counter were used, this would result in only values from 0 to $2^{15} - 1$ (32 768 ADU) instead of a possible range of 0 to $2^{16} - 1$ (65 536 ADU), thereby reducing the amount of possible values by half. Bias values typically range from a few hundred to a few thousand ADU.

[2]This could occur either through an electron from the conduction band dropping into the valence band, or measurement errors in the ADC.

Readout Noise:
There are three main noise sources in detectors: shot noise, fixed pattern noise, and read noise. Shot noise refers to the quantum characteristics of coherent photon states resulting in a Poisson distribution. This effect can be reduced by taking multiple images. Fixed pattern noise describes the behavior of individual pixels. This effect can be removed by subtracting bias frames (zero-second exposed images).

Readout noise results from the amplification stage of the sensor's readout. The quantification of a continuous signal leads to a statistical scatter. If this is the dominant source of noise, a longer exposure is advisable. The total effective noise (σ_{eff}) can be described by Eq. 3.7 with the dark shot noise (σ_D), read noise (σ_R), photon shot noise (σ_S), and fixed pattern noise (σ_F):

$$\sigma_{eff} = \sqrt{\sigma_D^2 + \sigma_R^2 + \sigma_S^2 + \sigma_F^2} \tag{3.7}$$

Dark Current:
A sensor cannot differentiate between an electron that originated from the silicon being struck by a photon, and exterior electrons (for example: leaked from the electronics of the camera, or thermal energy). This signal is strongly correlated with the detector's operating temperature and is described by Eq. 3.8, the Meyer-Neldel rule (based off the Arrhenius law of reaction rates proportionality to temperature (Clark, 2013)):

$$X = X_0 \cdot \exp\left[-\frac{E_A}{kT}\right] \tag{3.8}$$

Here we have the activation energy E_A, the absolute temperature T, the Boltzman constant k, and a factor dependent on the activation energy X_0. This noise is also dependent on time, and in modern sensors, with suitable cooling and non-excessive exposure times, can be negligible. To remove this unwanted signal, a non-illuminated image (or, more commonly, the mean of multiple images), exposed for the same time as the science frame can be subtracted from the original image.

Gain / Dynamic range

Gain is the term that refers to the uncertainty of quantification at the ADC and describes how many electrons are needed to produce one ADU step. The minimal gain for a sensor is therefore given by the bit-counter of the ADC and the full-well depth of the sensor's pixels. If we have, for example, a 14-bit ADC (max 16 383 ADU) and a pixel full-well depth of 100 000 electrons, the minimal gain would be (Eq. 3.9):

$$gain = \frac{\text{Max nr. of electrons per pixel}}{\text{Max nr. of counts per pixel}} = \frac{100\,000\, e^-}{16\,383\, ADU} = 6.1\, \frac{e^-}{ADU} \tag{3.9}$$

In some cases, a higher gain than the theoretical minimum can have certain advantages. If the overall noise of a CCD where at ± 10 electrons, then two values with a difference of 6 electrons don't contain any meaningful information, and it would be okay to assign them both the same ADU value[3] (Howell, 2006). If the gain is too large though, the dynamic range can be strongly impacted.

The *dynamic range* describes how many different levels of signal can be depicted (i.e. the ability to detect both dim and bright objects on the same image). Lets assume two detectors with max 1 000 electrons and a gain of 5 and 50 e^-/ADU respectively. The first detector would have 200 brightness levels to depict the signal, whereas the second would only have 20 discrete levels. This would mean, any signal between 850 and 900 electrons would result in the same brightness, whereas the first detector could differentiate between 10 varying levels in the same range.

Linearity:

The linearity of a detector describes the pixel's response to incoming photons as its potential well fills up. This means that the proportion of electrons per photon should stay the same no matter how full the pixel is. However, linearity typically decreases as a pixel saturates and it's potential well becomes less responsive.

[3] This uncertainty is known as the differential nonlinearity (DNL), and is most commonly chosen as ± 0.5 ADU.

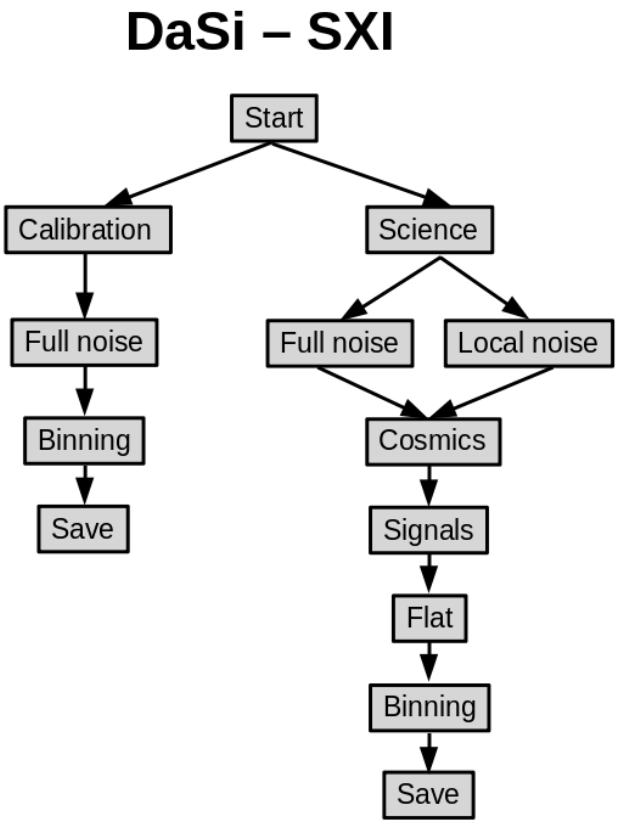

Figure 3.6: *Work Flow of DaSi. This modular layout of the simulator allows for the ease of adding new functions for future missions.*

3.2 The Simulator

The SMILE data simulator (DaSi) is an object oriented simulator and was developed in Python. The current version (`1.40`) is split into 6 main programs:

- A *configuration file* [`config.xml`] for defining global properties such as detector dimensions.
- A *constants file* [`constants.py`] for creating a library of the variables set in the config file.
- A *core-library file* [`core_lib.py`] for defining functions for the simulator object.

- A *global-library file* [`global_lib.py`] for linking the config.xml file to the simulator.
- A *utilities file* [`utilities.py`] for external functions not directly related to the simulation.
- And a *main file* [`main_prog.py`] for the simulation itself.

In the full data chain, the simulator is called by the FEE (see Sec. 5.1) with four parameters: exposure time, ccd number, calibration, and binning. The main work flow of DaSi is depicted in Fig. 3.6.

3.2.1 Configuration

The configuration file `config.xml` includes metadata such as general simulation information, graphics settings, detector settings, and signal information. A list of these configurations can be found in Tab. 3.1-3.2. These settings are called by the `get_xml_file` function in the `global_lib` file, and parsed into a `configuration` object as defined by `sim_config` in the `constants` file. The file is expected to grow as mode modules are added to the simulator.

Here, the main characteristics of the simulation can be defined by the user. It is important to keep in mind, that when changing these values, they stay in the same data type as defined in the global library. If the simulator is not called by the FEE, a number of runs can be defined which will then be save as a fits cube. This parameter was implemented for the ability to create a specific set of data or scenarios needed for testing. More to the creation of the fits-cubes can be found in Sec. 3.2.4. The setting *full_noise* can be set to either create a full frame of noise (0), or create noise just around the generated events and cosmics (1). Note that a full frame of noise can potentially result in run time longer than the expected typical exposure time depending on the system running the simulation. Standard binning is set to 6 (representing 6x6 binning) but this value will be overwritten if given other parameters by the FEE. Parameters for *single detector* dimensions are also set here, in the case that multiple detectors are present when creating one image. Cosmic and event values are generally empirically set to generate results similar to scientific solutions (as described in Sec. 3.1.1).

Setting	Default Value	Data Type	Description
General			
runs	1	int	Amount of runs per call (for scenarios)
Graphics			
show_plot	0 [False]	str	Show image of result
save_plot	0 [False]	str	Save image of result
save_data	0 [False]	str	Save data as .dat file
Detector			
detector_x	4510	int	x (horizontal) dimension of detector
detector_y	7582	int	y (vertical) dimension of detector
bias_level	150	float	Mean bias value in ADU
read_noise	4.5	float	Mean read noise in ADU
dark_current	0.0	float	Mean dark current in ADU
node_offset	0.1	float	Offset caused by each detectors two nodes
full_noise	1	int	Make full (0) or local (1) noise
binning_set	1 [True]	str	Set whether to bin or not
binning	6	int	Binning (for both x and y direction)
...			

Table 3.1: *a. DaSi configuration settings and default values.*

Setting	Default Value	Data Type	Description
...			
single_det_x	4510	int	x dimension for case of multiple CCDs
single_det_y	3791	int	y dimension for case of multiple CCDs
Cosmics			
cosmic_folder	cosmics_data	str	Folder holding cosmics data
cosmic_size	1	float	Size of cosmic relative to sensor
cosmic_intensity	5000	float	Mean intensity of cosmics in ADU
cosmic_rate	124	float	Mean amount of cosmics per frame
Signal			
event_amount	400	int	Mean amount of events per frame
event_intensity	100	float	Mean intensity of events in ADU
event_variability	2	float	Variability of amount of events per frame

Table 3.2: *b. DaSi configuration settings and default values continued.*

3.2.2 Coding a detector

Once called, the simulator first checks whether a calibration frame or a science frame is requested.

Calibration frame:
In the case of a calibration frame, only the noise and binning is executed before returning the array.

```
if callib != 0:
  frames = [datasim.callibframe(ccdnr, datasim, t1b, binningN,\
            const.sim_config[bias_level],\
            const.sim_config[dark_current],\
            const.sim_config[read_noise])]
  block  = save_blocker(block, frames, frames, len_x, len_y)
```

Here, the dimensions are set to a binnable length, the frame is initialized, the noise is added, and finally the frame is binned:

```
def callibframe(self,ccdnr,datasim,t1,binningN,bias,dc,rn):
  frame,(len_y,len_x) = self.calib_maker(self.single_det_x,\
                        self.single_det_y, binningN, bias, rn, dc)
  frame               = utils.rebin(frame,(int(len_y/binningN),\
                        int(len_x/binningN)))
  return frame
```

with

```
def calib_maker(self,x,y,binning,bias,rn,dc,node_offset1,    \
                node_offset2,noise_type):
  len_x   = utils.set_detect_len_to_bin(x,binning)
  len_y   = utils.set_detect_len_to_bin(y,binning)

  frame_1 = self.types_of_noise(noise_type,bias,dc,rn,len_x, \
                                len_y)
```

```

    frame_1[0:len_x,0:int(len_y/2)]        *= (1+node_offset1)
    frame_1[0:len_x,int(len_y/2):len_y*2] *= (1+node_offset2)

    frame_1 = np.clip(frame_1,0,2**16-1)
    return frame_1,np.shape(frame_1)
```

and

```
def types_of_noise(self,type_noise,bias,dc,rn,len_x,len_y):
    u1,u2 = [],[]
    #0 = np.normal, 1 = np.rand+BM, 2 = cv2.rand+BM
    if type_noise == 0:
        frame_1   = np.reshape(np.random.normal(bias+dc, rn,\
                                len_x*len_y), (len_y,len_x))

    elif type_noise == 1:
        u1        = np.random.rand(len_y*len_x)
        u2        = np.random.rand(len_y*len_x)
        z1        = self.gaussian(u1,u2)
        frame_1   = np.reshape(rn*z1+(bias+dc),(len_y,len_x))

    elif type_noise == 2:
        mat       = np.zeros(len_y*len_x)
        list_rand = cv2.randn(mat,(bias+dc),rn)
        frame_1   = np.reshape(list_rand,(len_y,len_x))

    return frame_1
```

The random (gaussian) numbers for the noise can be generated in three different versions which are defined by the user (see Sec. 3.2.3):

```
def gaussian(u1,u2):
    z1 = np.sqrt(-2*np.log(u1))*np.cos(2*np.pi*u2)
    return z1
```

Science Frame:

Initialize:

If a full frame is requested, the algorithm first checks what type of frame is to be made (Options: CCD_A, CCD_B, or CCD_A+B):

```
else:
  ### INITIALIZE FRAME: ###<
  frame_1,frame_2=[],[]
  if   ccdnr == 'CCD_A'  : frames = [frame_1]
  elif ccdnr == 'CCD_B'  : frames = [frame_2]
  elif ccdnr == 'CCD_A+B': frames = [frame_1,frame_2]
```

and initializes the frame(s):

```
frames,len_x,len_y = datasim.initialize_frame(len(frames),binningN)
```

with:

```
"""
INITIALIZE:
- sets detector size to binnable length
- creates two empty frames
"""
def initialize_frame(self,nrs,binning):
  #throws away undivisible rows/columns
  len_x   = utils.set_detect_len_to_bin(self.single_det_x,binning)
  len_y   = utils.set_detect_len_to_bin(self.single_det_y,binning)
  det_dim = (len_x,len_y)
  frames  = np.zeros((nrs,len_x,len_y))
  for i in range(nrs):
    frames[i] = np.empty(det_dim)
  return frames, len_x,len_y
```

Full Noise:

From here, the program checks if full-frame noise is to be added (`full_noise = 0`) or just locally at events (`full_noise = 1`). For full noise:

```
### CREATE FULL NOISE FRAME ###
if fullnoise == 0:
  frames = datasim.make_full_bias(const.sim_config[bias_level], \
           const.sim_config[dark_current],                      \
           const.sim_config[read_noise],node_offset,frames)
```

which calls:

```
1  """
2  FULL BIAS GENERATOR:
3  - introduces bias-value/readnoise, clips to non-negatives
4  - generates full gaussian bias/RN (dark current pending)
5  """
6  def make_full_bias(self,bias,dc,rn,node_offset,frame,noise_type):
7    len_x,len_y = np.shape(frame[0])
8    for i in range(len(frame)):
9      frame[i] = self.types_of_noise(noise_type,bias,dc,rn,len_y,   \
10                                    len_x)
11     frame[i] = np.clip(frame[i],0,2**16-1)
12     frame[i][0:len_x,0:int(len_y/2)]       *= (1+node_offset[2*i])
13     frame[i][0:len_x,int(len_y/2):len_y*2] *= (1+node_offset[2*i+1])
14   return frame
```

Here, in l. 14, the noise with the mean of bias and dark current values and deviation of the read noise is distributed across the whole detector(s), and subsequently clipped from 0 to $2^{16} - 1$ to avoid negative values or values higher than what the bit-counter would register (a 16-bit ADC is used, meaning values above $2^{16} - 1$ would be a saturated pixel). Any value below 0 or above $2^{16} - 1$ is set to that respective value. Next, an offset is added to each half of the frame, to account for a slight node offset (l. 17-18). The values for the node-offset where taken from standard-frames released by Open University.

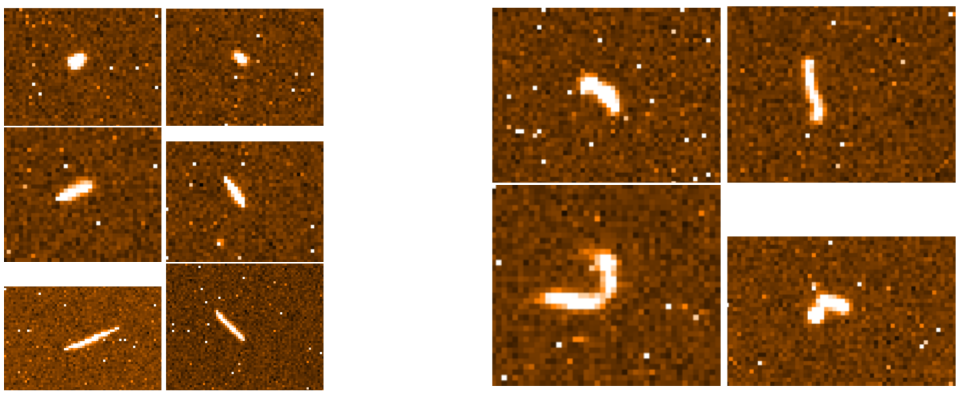

Figure 3.7: *Examples of cosmic rays.* ***Left:*** *muons.* ***Right:*** *worms (Bremer and Nobels, 2015).*

Cosmics:
Cosmic rays are high energy particles originating from the sun, sources outside of the solar system, and from distant galaxies, which produce bright spots on an image. The cosmics are generated and added to the array:

```
### SHOOT COSMICS ###
frames = datasim.place_cosmics(frames , ... )
```

`place_cosmics` first calculates an amount of cosmics to be placed (l. 16). It then calculates for each cosmic: a location (Frame A or B); intensity; orientation; and type of cosmic (l. 20-30). The two general types of cosmics are defined as straight/oval *muons* (70% probability) and the more elliptical *worms* (30% probability). Examples of observed muons and worms can be seen in Figs. 3.7.

```
"""
COSMIC GENERATOR:
- amount of cosmics = exp_t [s] * cosmic-rate [cosm/sec]
- chooses random array, rotation, file_nr
- chooses worm or muon (30/70 probability)
- loads cosmic-file (faster than generating on own)
- append cosmic to array
"""
```

```
11 def place_cosmics(self, frame, cosmic_folder, size, intensity, \
12                   exp_t, cos_rate, bias_level, dark_current,   \
13                   read_noise, binning, fullnoise, bias, dc,    \
14                   rn, node_offset):
15
16   amount     = np.random.randint(cos_rate*exp_t-10,cos_rate*exp_t+10)
17   for i in range(amount):
18
19     ### define placement/orientation/shape/type ###
20     side              = np.random.randint(0,len(frame))
21     intensity         = np.random.randint(intensity-50,intensity+50)
22     rotate            = random.randint(1,4)
23     nr                = str(random.randint(1,99))
24
25     #worms = 30%, muons=70% prob
26     cos_type          = random.randint(1, 100)
27     if cos_type < 31:
28       name          = 'w'
29     else:
30       name          = 'c'
```

Next, the cosmic files are loaded[4]. In the first version of the simulator, cosmics were generated from scratch, however, the computational cost needed to calculate and store completely unique cosmics each time went above what the resources allowed. To counter this, an inventory of precalculated cosmics was generated (of both worms and muons), which are then called by the program. The end shape of the cosmics are given further variation by the binning, since not the exact same 6×6 pixels of a cosmic will be binned together. The behavior and generation of cosmics was adapted from Bremer and Nobels (2015). The csv-file is read and appended to an array `cosmic_dat` (l. 32-39), next the cosmic is increased and rotated (l. 42-47), and a location on the detector is generated (l. 54-57):

[4]In the Raspberry-Pi version RAPIDsim, these cosmics are all loaded on initialization (see Sec. 5.3).

```
### open cosmics ###
cosmic_dat      = []
cosmic_filename = cosmic_folder+name+nr+'.csv'
with open(cosmic_filename, "r") as csvfile:
  lines  = csvfile.readlines()
  for line in lines:
    cosmic_dat.append(line.split(","))
#removes line split "\n"
cosmic_dat = [line[:-1] for line in cosmic_dat]

#increase size in both directions
cosmic_dat = np.repeat(cosmic_dat, size, axis=1)
cosmic_dat = np.repeat(cosmic_dat, size, axis=0)

#rotates cosmics
for i in range(0,rotate):
  cosmic_dat  = np.rot90(cosmic_dat)

cosmic_dat      = np.asarray(cosmic_dat, dtype='float64')
cosmic_dat     *= intensity

fheight, fwidth = frame[side].shape
cheight, cwidth = cosmic_dat.shape
pos_x           = np.random.randint(0,fwidth)
pos_y           = np.random.randint(0,fheight)
pos_x2          = pos_x + cwidth
pos_y2          = pos_y + cheight
```

Before the cosmic is added to the frame, a buffer (`extend`) is generated, with the buffer size based off the binning value (l. 58-61). This buffer is used to keep both the cosmics inside the frames, as well as for adding the needed local noise:

```
58 if binning < 10:
59   extend = 40
60 else:
61   extend = 100
62
63 hb,wb    = cheight+extend,cwidth+extend
64 pos_x3   = pos_x+wb
65 pos_y3   = pos_y+hb
66
67 #check for possible index out of range
68 if pos_x3 > fwidth or pos_y3 > fheight:
69   continue
```

Assuming the cosmic is inside the frame, if the image wasn't given a full-frame noise, the local noise around the cosmic is generated and added to the cosmic (including the buffer area), and the cosmic is added to the frame:

```
70     else:
71       try:
72         if fullnoise != 0:
73           frame = self.make_local_noise(frame, side, pos_x, pos_x2,\
74                   pos_y, pos_y2, fwidth, node_offset, extend, hb,  \
75                   wb, bias, dc, rn)
76         frame[side][pos_y:pos_y2, pos_x:pos_x2] += cosmic_dat
77       except:
78           pass
79 return frame
```

The local noise is defined similar to the full noise, only additionally accounting for the node offsets:

```
"""
LOCAL NOISE GENERATOR:
- introduces bias-value/readnoise, clips to non-negatives
- generates full gaussian bias/RN (dark current pending)
"""
def make_local_noise(self,frame,side,pos_x,pos_x2,pos_y,pos_y2, \
                     fwidth,node_offset,extend,hb,wb,bias,dc,rn):

  # where zero, stays zero!
  biasarr_mask = np.where(frame[side]\
                 [int(pos_y-(extend/2)):\int(pos_y2+(extend/2)),\
                  int(pos_x-(extend/2)):int(pos_x2+(extend/2))] \
                  == 0,1,0)
  if   side == 0 and pos_x < fwidth/2: nodeval  = node_offset[0]
  elif side == 0 and pos_x > fwidth/2: nodeval  = node_offset[1]
  elif side == 1 and pos_x < fwidth/2: nodeval  = node_offset[2]
  elif side == 1 and pos_x > fwidth/2: nodeval  = node_offset[3]

  biasarr   = self.types_of_noise(noise_type,bias,dc,rn,hb,wb)
  biasarr   = np.swapaxes(biasarr,0,1)

  if np.shape(biasarr_mask)[1]==0 or np.shape(biasarr_mask)[0]==0:
    return frame

  biasarr = np.clip(biasarr_mask*(np.add(biasarr,nodeval)),1,2**16-1)
  frame[side][int(pos_y-(extend/2)):int(pos_y2+(extend/2)), \
              int(pos_x-(extend/2)):int(pos_x2+(extend/2))] \
              += biasarr
  return frame
```

	Sim 1 - low activity		Sim 2 - high activity	
	Counts $cm^{-1}s^{-1}$	Counts $CCD^{-1}s^{-1}$	Counts $cm^{-1}s^{-1}$	Counts $CCD^{-1}s^{-1}$
SWCX Rate	0.093	5.134	1.483	82.150
SXRB Rate	0.278	15.403	0.278	15.403
Particle Bkg Rate	0.011	0.627	0.023	1.255
Mean Point Source Rate	0.012	0.657	0.012	0.657

Table 3.3: *Simulated SWCX flux and rates for two cases (Soman and Randalls, 2018).*

Here, a mask is created with `numpy.where`. This masks all locations that are not equal to zero (l. 10-13). The mask prevents an accumulation of noise values if, for example, the buffer section of two or more cosmics/events overlap. Essentially, if a non-zero value is already at a pixel, the mask is set to zero, and if not the mask is set to one, then the noise array is multiplied by this mask. Thereby ensuring no overlap is added. Then, the bias, dark current, read noise, and node-offset are added and the image is masked and clipped just as in the full-frame noise (l. 14-25). The created bias-array is added to the cosmic-array and returned (l. 26-28).

Events:
The information for expected events (size, frequency, etc.) where taken from Soman and Randalls (2018). X-ray events (of energy ranging from 0.5 - 1.1 keV) will be imaged on 108 μm square pixels (6×6 binned). The SXI will have a FoV of 15.5 × 26.5° with a total CCD area of 8.12 cm × 6.82 cm × 2 CCDs. A list of SWCX rates can be seen in Tab. 3.3.

For an extreme case, these values can be doubled, this would lead to a maximum case of 328.6 events per full 2 CCD frame per second (Eq. 3.10):

$$82.15\ events\ CCD^{-1}\ s^{-1} \cdot 2\ CCDs \cdot 2 = 328.6 \text{ events s}^{-1} \text{ in full 2 CCD frame} \quad (3.10)$$

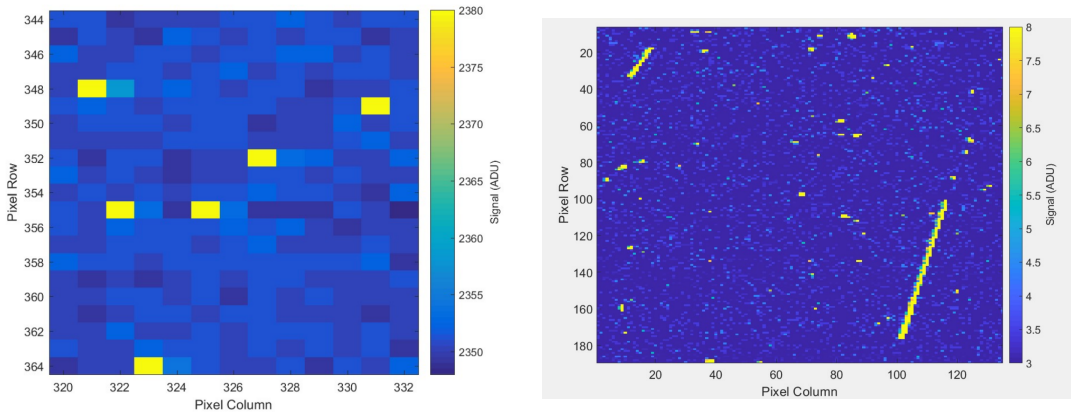

Figure 3.8: ***Left:*** *X-ray events imaged by PLATO CCD270.* ***Right:*** *Example including cosmic hits. Yellow pixels represent hits (Soman and Randalls, 2018).*

The cosmic ray flux is estimated to be about 2.25 events $cm^{-1}s^{-1}$, meaning for a one second 2 CCD exposure, around 249 cosmics can be expected, which is double that of the expected normal SWCX event rate. An example image of events and cosmics can be seen in Fig. 3.8. Soft X-ray background events (SXRBs) as well as the particle background (PB) also need to be taken into account, which are not discernible from science X-rays and leads to an additional 69.26 X-ray events per second per 2 CCD image (Eq. 3.11):

$$(15.403+1.255+0.657)\ counts\ s^{-1}\cdot 2\ CCDs\cdot 2 = 69.26 \text{ Bkg X-rays } s^{-1} \text{ per 2 CCD image} \tag{3.11}$$

Due to binning, and the size of the pixels in relation to the size of X-ray events, only a few sets of shapes are possible. These can be seen in Fig. 3.9. Approximately 99.87% of events are single pixel hits, 0.064% are "doubles" (types 1-4), and the rest are types 5-31.

The events are generated in a similar fashion as the cosmics:

```
### SHOOT EVENTS ###
frames      = datasim.place_signal(frames,  ... )
```

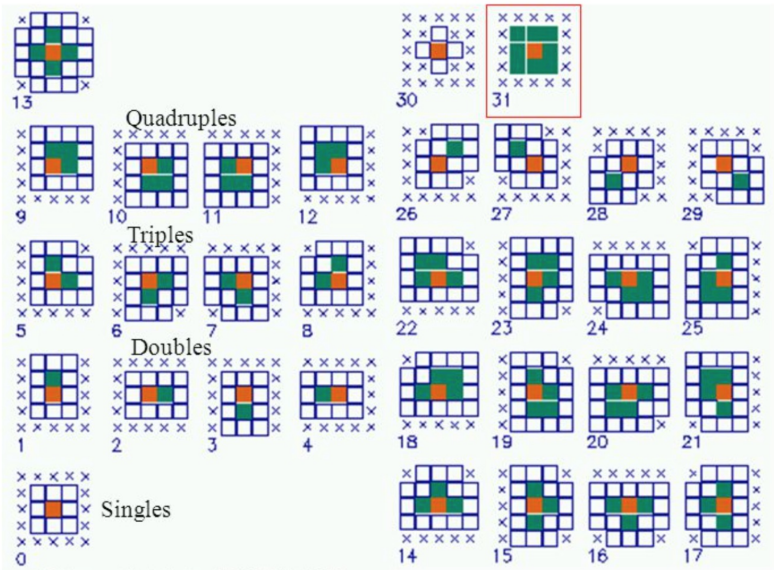

Figure 3.9: *Possible event shapes. Approximately 99.87% of events are single pixel hits. 0.064% are doubles (types 1 - 4) (Read, 2018).*

with

```
"""
SIGNAL GENERATOR:
"""
def place_signal(self,frame,amount,intense,var,fullnoise,node_offset,\
                 bias,dc,rn,binning):
  x,y      = np.shape(frame[0])
  amount   = np.clip(int(np.random.normal(amount,var,1)),0,2*amount)

  for i in range(amount):
    xs1,ys1        = np.random.randint(3,x-3),np.random.randint(3,y-3)
    side           = np.random.randint(0,2)
    rotate         = np.random.randint(1,4)
    signal_nr      = np.random.randint(0,len(a))
    intens         = np.random.randint(intense-var,intense+var)
    sig            = np.multiply(np.reshape(a[signal_nr],[4,4]),intens)
    sig            = cv2.flip(sig,xx)
```

```
17
18      if binning < 10:
19        extend = 40
20      else:
21        extend = 100
22      hb,wb    = 4+extend,4+extend
23
24      try:
25        if fullnoise != 0:
26          frame = self.make_local_noise(frame, side, ys1-2,     \
27                  ys1+2, xs1-2, xs1+2, y, node_offset, extend, \
28                  hb, wb, bias, dc, rn)
29        frame[side][xs1-2:xs1+2, ys1-2:ys1+2] += sig
30        except:
31          pass
32    return frame
```

Again, an amount of events is generated, and as with the cosmics, an inventory of events was generated to be used. It is also important to note that, due to the binning, these shapes also change depending on their location on the detector. Then for each event a x- and y-position, side of the detector, rotation, type of signal, and intensity are calculated (l. 10-16). The signal is given a buffer (l. 18-22), and noise is added in the local case (l. 25-28) and the event is added to the frame (l. 29). A `try/except` clause is added in the case that something malfunctions (approximately every 1 000th event returns an error in the masking-stage.)

Flats:

Lastly, the flats are generated. Flat fields are effects of both pixel-to-pixel sensitivity variations, as well as distortions brought on by the optical path. Flat field corrections are done by dividing a science frame by a normed (to 1) flat field image (an image of an illuminated flat surface). The flat was created by digitizing images from Read (2018) (see Fig. 3.10).

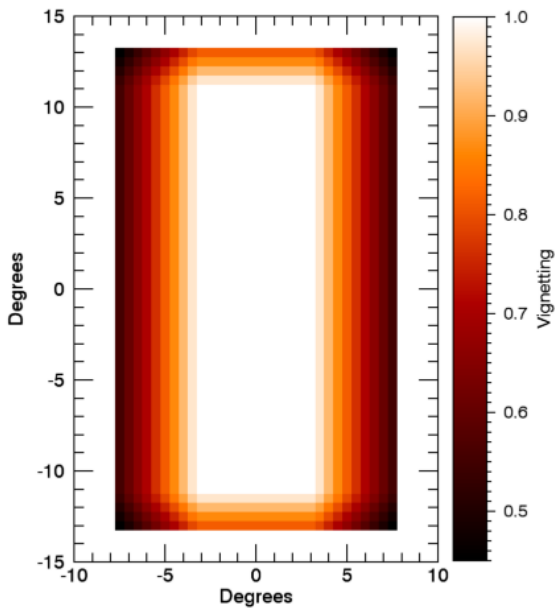

Figure 3.10: *Simulated flat field of SXI (Read, 2018).*

The flat field is called with:

```
### FLAT ###
frames = datasim.generate_flat(frames,'flat_self.csv', len_x, len_y)
```

with

```
"""
FLAT GENERATOR:
- reads flat.csv file, resized to size of both arrays
- blockshaped: returns array blocked into two halves
- multiply arrays by respective flat side
"""
def generate_flat(self,frame, flat_file,len_x,len_y):
  array1          = frame[0]
  flat_data       = pd.read_csv(flat_file, quoting=csv.QUOTE_NONE)
  flat_data       = np.array(flat_data)
  x,y             = np.shape(array1)
  xs,ys           = np.shape(flat_data)
```

```
15    flat_normed    = cv2.resize(flat_data,None,fx=2*y/ys, fy=x/xs,\
16                     interpolation = cv2.INTER_CUBIC)
17    x,y            = flat_normed.shape
18    flat_block     = utils.blockshaped(flat_normed,int(x),int(y/2))
19    for i in range(len(frame)):
20      frame[i]       = np.multiply(frame[i],flat_block[len(frame)-1-i])
21    return frame
```

The csv-file of the flat file is imported and resized to the size of the (full 2 CCD) detector (l. 10-16). Resizing is done with cubic interpolation using the `opencv [cv2]` package. With the function `blockshaped` (l. 18), the flat-array is split into two separate arrays (for each detector side)[5]. And finally, the science frame is multiplied by the flat frame (l. 19-20). The benefit of generating the flat this way is that an arbitrary flat-field can be applied by simply defining a csv file. The `blockshaped` function is as follows:

```
def blockshaped(arr, nrows, ncols):

  h, w = arr.shape
  return (arr.reshape(h//nrows, nrows, -1, ncols).swapaxes(1,2)\
          .reshape(-1, nrows, ncols))
```

The final step, if specified, is to bin the array:

```
### BIN FRAMES AND COMBINE ###
framer = []
if binningN > 1:
  for i in range(len(frames)):
    framer.append(utils.rebin(frames[i],(int(len_x/binningN),\
                  int(len_y/binningN))))
          xx,yy  = np.shape(framer[0])
  framer = np.clip(framer,0,2**16-1)
```

[5]Note, in the RAPIDsim version, this is also all done only once at initialization (see Sec. 5.3).

The array `framer` is used as a placeholder for the binned frames. After rebinning, the entire frame is once again clipped to values between 0 and $2^{16} - 1$ so as to avoid "impossible" values that couldn't be generated by the ADC.

The `rebin` function is:

```
def rebin(array1,new_shape):
  shape1 = (new_shape[0],array1.shape[0]//new_shape[0],\
            new_shape[1],array1.shape[1]//new_shape[1])
  return array1.reshape(shape1).sum(-1).sum(1)
```

This method was created with the helpful folks at *stackoverflow*. The precise method of how it works will be explained in the following chapter (Sec. 3.2.3).

The frame can then be saved, or is simply returned to the requesting socket. Saving the simulated data will create a .FITS file, which includes a header of the user-defined settings (more in Sec. 3.2.4).

An example full frame (both CCD sides with full noise and flat) can be seen in Fig. 3.11.

3.2.3 Optimization

The simulator is aimed to be able to be implemented in real time into the SXI simulator. This means that the time from the moment an image is requested, to when it needs to be delivered to the FEE SW simulator program is 4 seconds (TBC). If we assumed that we only needed to calculate single values for all ~35 million pixels (assuming a full frame of 4510×7582 pixels), a maximum of 170 nanoseconds is allowed per pixel. One single calculation in Python generally takes around 1 500 cycles (for an i7 8th-gen processor running at base speed of 1.7 GHz, established using the `hwcounter` library).

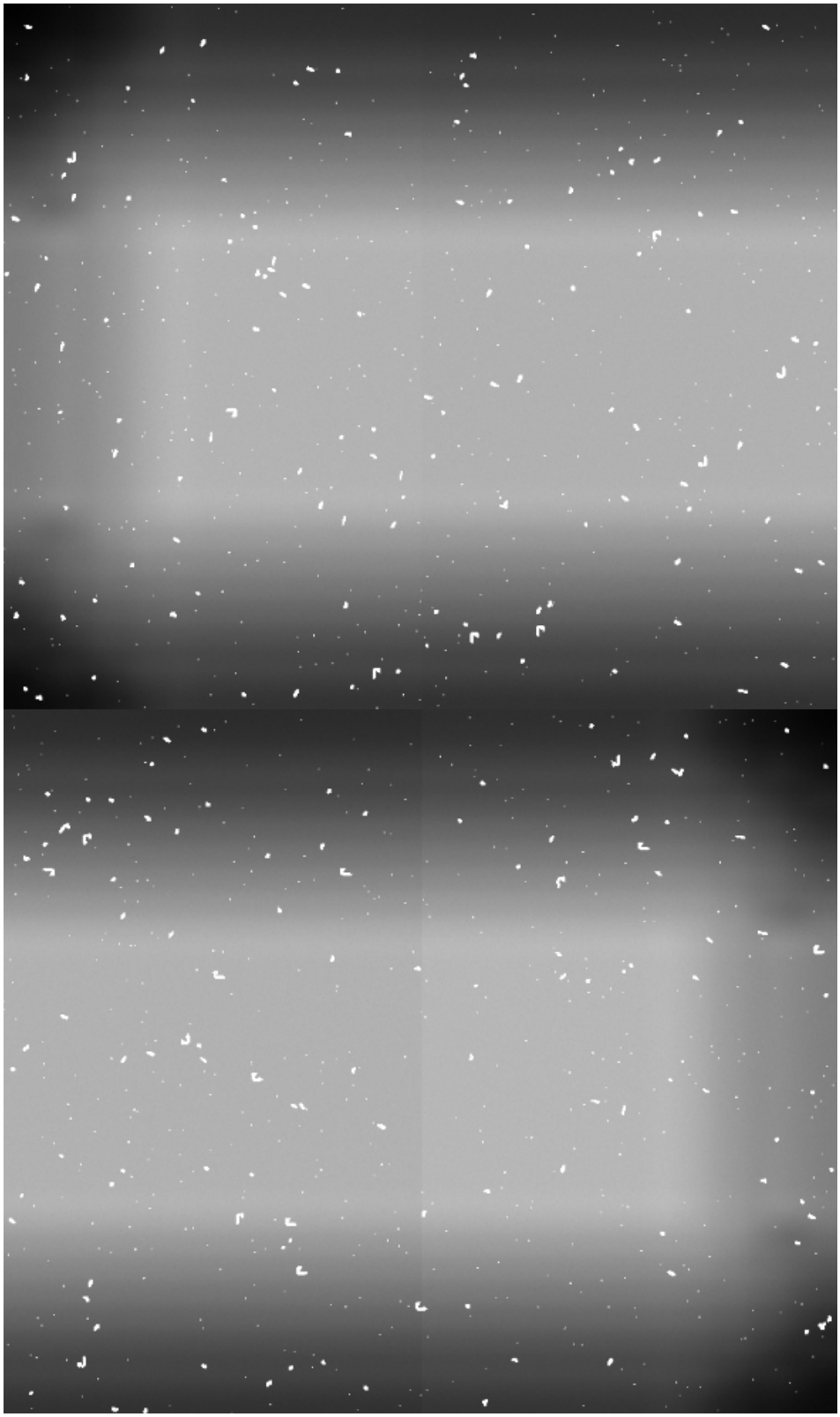

Figure 3.11: *Sample full noise, full frame image of DaSi simulation. CCD_A is the top half and CCD_B the bottom half. Since the passing of the data to the FEE is serialized, this is how the data will be read. The events are the small grey areas, and cosmics the bright elongated sources. The node offsets can be seen in each half of a CCD frame.*

If we extrapolate this, that would mean:

1 500 cycles · 35e9 pixels = 51 billion cycles

for *one* calculation for the calculation of a single exposure. Which in turn implies:

5.1e10 / 1.7e9 = 30 seconds

for an entire frame, and even at a turbo boost frequency of 4.4 GHz, a time of 12 seconds would be needed. Obviously, the math is not this simple when applying it to processors, and this changes once more when handling multiple values at the same time in Python. However, the problem with computational resources is clear, especially when taking into account that more than one calculation is required per pixel. And in fact, after the initial simulator was complete, a single run would take between 17 and 25 seconds.

Each step of the simulation was measured to isolate the functions most grievously in need of resources. The functions found to require the most time were the generation of noise, and the binning of the image, with both taking 6 and 4 seconds respectively.

Random Number Generation:

Generating random numbers was done using the `numpy` library. `numpy` is already written in C (wrapped by CPython) and is made to be highly efficient. Comparing the `numpy` with Pythons standard `random` library, `numpy` was found to be over 10 times faster at creating gaussian-distributed random values (see Fig 3.12). An attempt was made at first to write a random number generator in C specifically for this simulator, but all attempts still ended up being slower than equivalent methods in `numpy`.

The seed for random number generation was generated via taking the last digits of the CPU's current clock cycle. The first steps to increasing efficiency of noise generation were to remove loops, replacing them with either list comprehension or other existing library functions that proved to already be optimized (such as Python's `reshape` module). Other options that were pursued were using quicker, less truly random methods

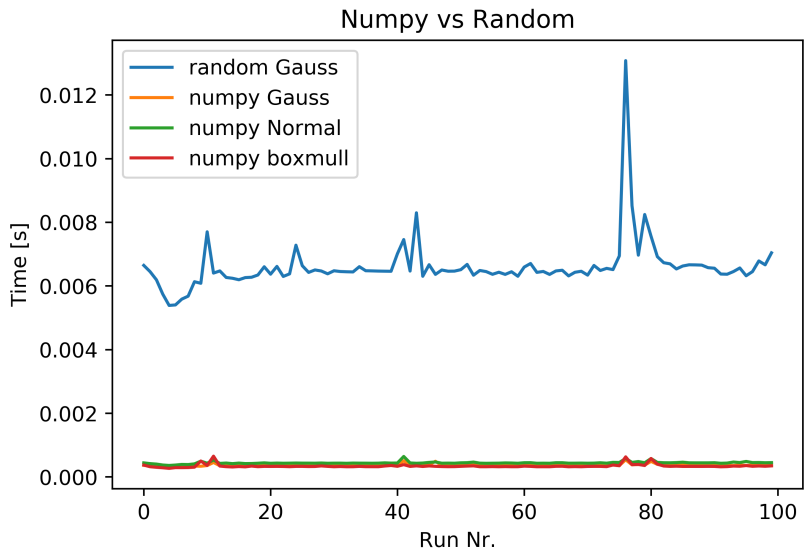

Figure 3.12: *A comparison of the* `random` *and* `numpy` *random value generators over 100 runs for 10 000 values per run per method. The* `random` *library is consistently over 10 times slower at generating values.*

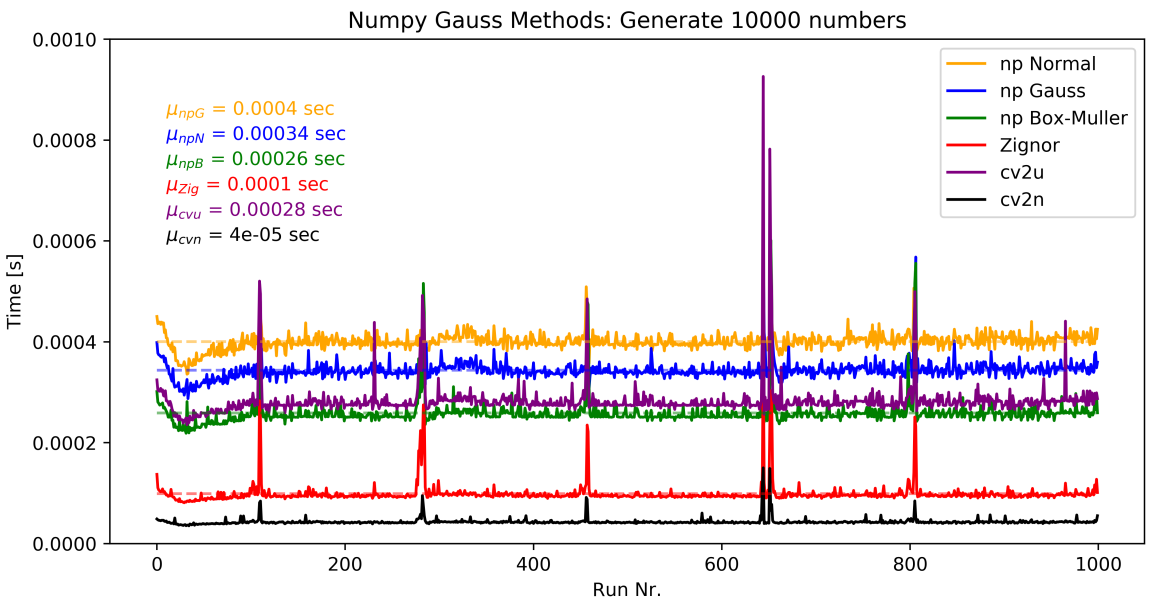

Figure 3.13: *A comparison of six different random-number generators. It is clear that the Uniform-Box-Muller transform, the Zignor, and cv2.Normal methods are most efficient. Note, Zignor is the mode name for the Ziggurat algorithm.*

of generating noise, such as creating a smaller subset of random noise and repeating that subset in random orders, or even reading the last digit of an operating systems value at random intervals (thereby foregoing any calculations). However, these where never significantly faster, and presented another issue. Since those values where never truly random, it was inevitable that certain patterns would emerge in the resulting noise. These patterns would be picked up by the data compression software and, for purposes of testing, result in compression ratios noticeably better than those if the noise were truly random.

The tested methods were:

(1) The numpy "standard-normal" distribution `numpy.random.randn` which generates gaussian distributed values with mean = 0 and standard deviation = 1.

(2) The numpy "normal" distribution `numpy.random.normal` which generates gaussian distributed values with a user defined mean and standard deviation.

(3) The numpy "uniform" distribution `numpy.random.uniform` which generates a uniform distribution with a user defined minimum and maximum value.

(4) The "Ziggurat" algorithm (rejection-based sampling).

(5) The opencv "uniform" distribution `cv2.randu` similar to numpy uniform.

(6) The opencv "normal" distribution `cv2.randn` similar to numpy normal.

The methods (1), (3), (4), and (5) where consequently sampled via the Box-Muller transform (Driss et al., 2018) to create a gaussian distribution of the same mean and standard deviation as defined by the configuration. The Ziggurat algorithm is based on rejection-sampling wherein a point is generated, and tested whether or not it is inside the desired distribution. It is very efficient yet complex to implement (Marsaglia and Tsang, 2000). A readily available implementation was uploaded on github by *James Lao*[6], and instead of reinventing the wheel, this library was used. The results of 1 000 test runs of generating 10 000 values can be seen in Fig. 3.13 and their histograms in Fig. 3.14. Methods (3), (4), and (6) where by far the most efficient.

[6]Download (incl. installation and usage) at `https://github.com/jameslao/zignor-python`.

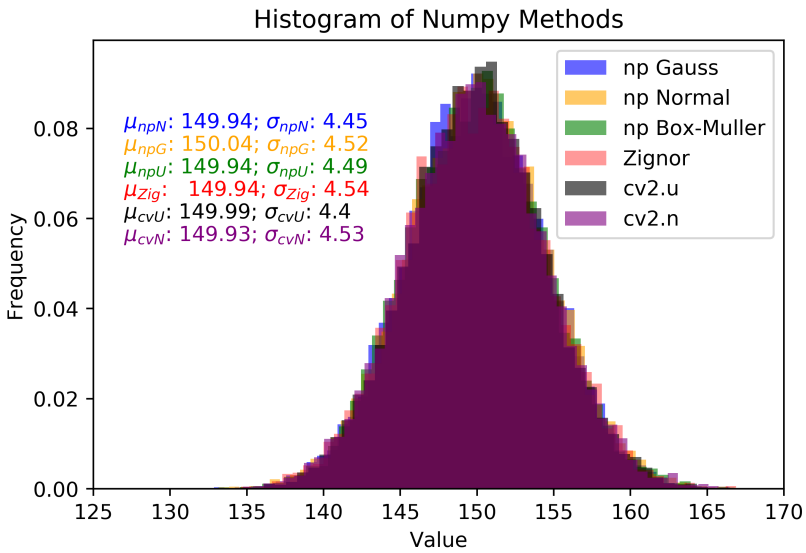

Figure 3.14: *The histograms of all six random-number generators. It is clear to see that both the mean, and standard deviation are the same for all cases.*

The numpy and opencv methods where chosen as the number generators. The opencv method is implemented as optional since it is not a native Python3 library and would need to be installed separately. The numpy uniform method of noise generation leads to simulations within the allotted run-time (see below), meaning the requirements are fulfilled. Also, this way no additional packages need to be downloaded to use the software. The performance of methods (2), (3), (4), and (6) when applied to DaSi can be seen in Fig. 3.15. Here, it is clear to see that the Ziggurat and opencv methods are ideal for both the full-noise science frame, and the calibration frame, with `numpy.random.uniform+BM` being the fastest of the numpy methods. Interesting is, that for the local noise, the Ziggurat and `numpy.random.gaussian` methods are nearly identical with only around 0.07 seconds difference. Also noteworthy is the fact that in this case, the Gaussian-distribution method is more efficient than the uniform + Box-Muller distribution method. In the case that numpy is chosen for number generation, the Gaussian method is used for local noise generation, and the normal method is used for full noise and calibration.

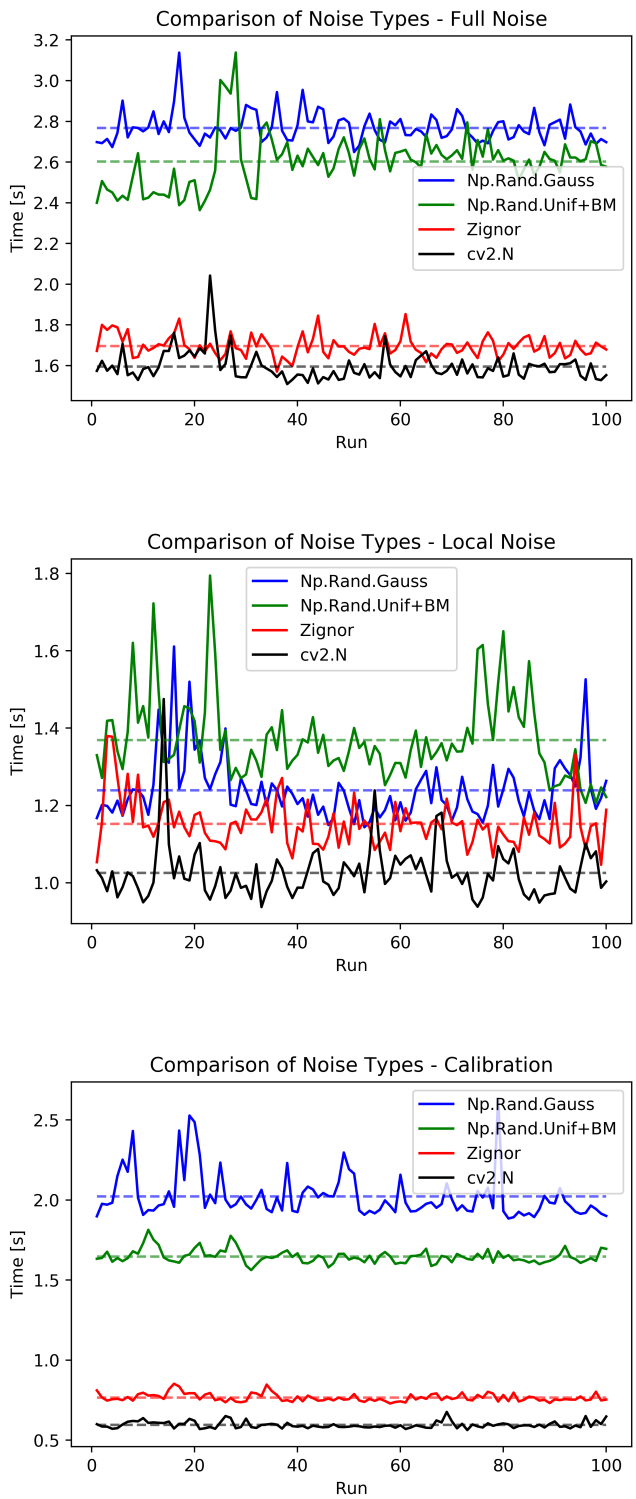

Figure 3.15: *Performance of the four methods* `numpy.random.gauss` *(**blue**),* `numpy.random.uniform+BM` *(**green**),* `Ziggurat` *(**red**), and* `cv2.normal` *(**black**) for all main three types of frame generation. **Top:** Full-noise science frame. **Middle:** Local-noise science frame. **Bottom:** Calibration frame.

Noise:

As previously established, detectors have a lot of sources of noise, each with their own origin and behavior. After the Box-Muller method was implemented, a full frame noise took on average 1.25 seconds. With the entire simulation taking on average 2.61 seconds, with a minimum of 2.52 and a maximum of 2.76 seconds. However, these simulations where typically run on a Lenovo ThinkPad outfitted with an intel i7 8th gen processor clocking up to 4.40 GHz and 16 GB of RAM. It was also tested on another machine with an i5 8th gen processor and 8 GB RAM and still stayed within the four second time limit. Despite this, it is best to keep in mind that if run on SMILE-like hardware (or, for testing purposes, on a raspberry pi module as described in Sec. 5.3), these times may increase above a real-time implementation.

Due to this, and also taking into account the fact that the FEE will only be looking for significant signals (refer to Sec. 5.2), an option to only generate local noise was implemented. Here, the initial frame skips the full noise generation, and during the cosmics and events phase, local noise is added in a buffer zone around the signal. As mentioned in the previous section, a mask is created to avoid overlapping noise intersecting events. Another step to optimization here could have been to check how many pixels fall away due to this mask, but since only a few values need to be generated, calculating the amount of pixels that fall away would take longer than the generation of those values. With this method, the simulation is consistently under 4 seconds (on average 1.05 seconds, with a minimum time of 0.98 and a maximum of 1.24 seconds), and the FEE source detection is not hindered since it first checks whether or not a significant signal occurred, and only then checks the "background" noise. A series of execution times over 100 runs with local and full noise can be seen in Fig. 3.16. A cutout of a frame with local noise can be seen in Fig. 3.17.

Binning:

Binning was originally set in a simple loop, were neighboring pixels where summed and placed in a new array. This obviously cost a lot of memory and computational space, especially when considering the fact that Python loops are notoriously bad at efficiency[7].

[7] Python loops pass the data to the interpreter every iteration, creating a substantial overhead.

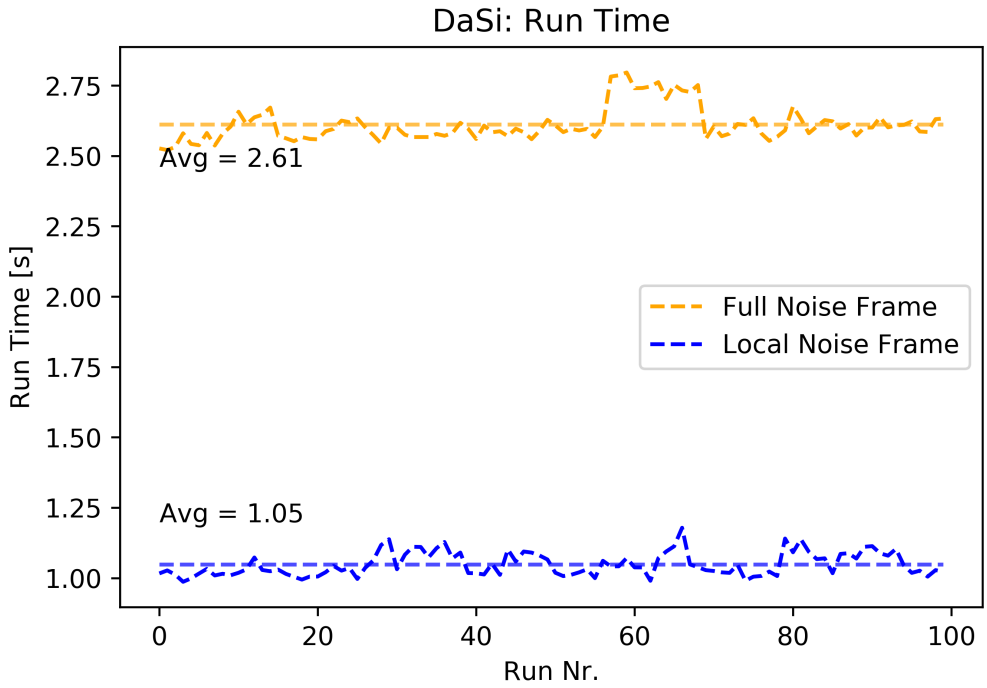

Figure 3.16: *Run time over 100 runs of both full frame and local noise methods.*

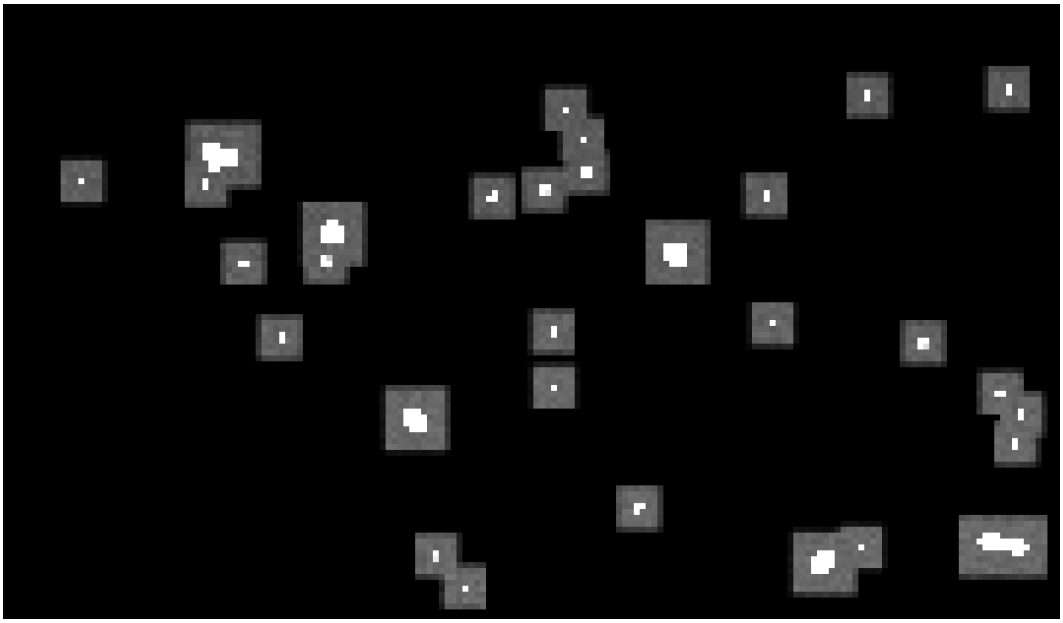

Figure 3.17: *Cutout of a DaSi frame generated with local noise around events and cosmics.*

Even after implementing an extension written in C of the same process, binning took over 2 seconds. Using the `map` function (which is essentially a C implemented version of a `for` loop), no substantial improvement was reached. After lots of late nights, a bit too much coffee, and lots of foresting around stackoverflow, a solution was found. The result was the code in the previous section. Note, this code also works for non-equal binnings.

To illustrate how these two beautiful lines of code work, lets look at a simple example of a 2×6 matrix being binned by 2×3 (so the result should be a 1×2 matrix). We take the matrix:

$$A = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 & 11 \end{bmatrix}$$

The function requires both the array itself (with dimensions `array.shape[0] = 2` and `array.shape[1] = 6`), and the new shape (which is the original dimensions divided by the binning value: `new_shape[0] = 2/2 = 1` and `new_shape[1] = 6/3 = 2`). First, the shape requirements are defined:

```
shape1 = (new_shape[0],array1.shape[0]//new_shape[0],\
new_shape[1],array1.shape[1]//new_shape[1])
```

which creates the shape `(1,2,2,3)`, 1 and the second 2 are the dimensions of the new binned array, and the first two 2 and the 3 are the binning values to the respective dimension. Next, the array is reshaped into sub arrays pairs of 3 for each line (`array1.reshape(shape1)`):

$$B_1 = \begin{matrix} [[[0 & 1 & 2] \\ [3 & 4 & 5]] \end{matrix} \qquad B_2 = \begin{matrix} [[6 & 7 & 8] \\ [9 & 10 & 11]]] \end{matrix}$$

Then, each pair gets summed along the "x-" or first-dimension (`.sum(-1)`) resulting in:

$$C_1 = [[3 \quad 12] \qquad C_2 = [21 \quad 30]]$$

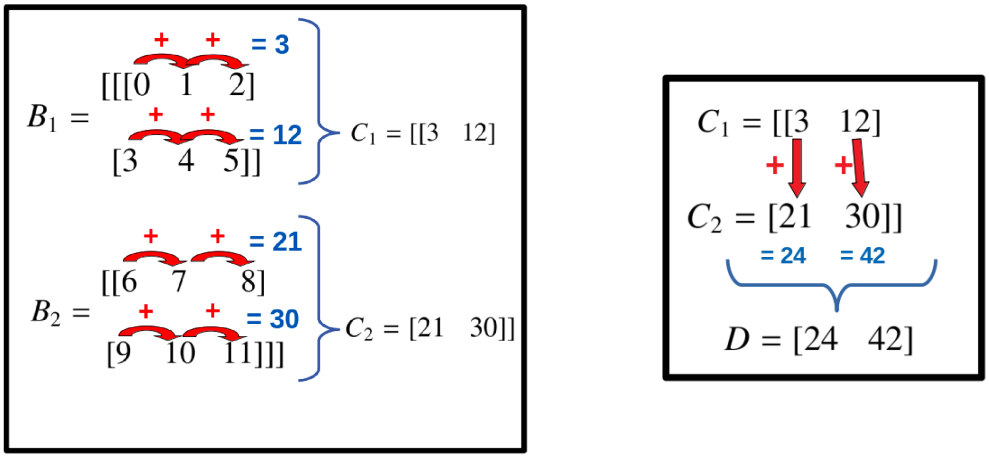

Figure 3.18: ***Left:*** *Summing in x-direction.* ***Right:*** *Summing in y-direction.*

And lastly summed along the "y-" or last-dimension (`.sum(1)`) resulting in:

$$D = [24 \quad 42]$$

This way, the entire array is passed through the interpreter once, calculated and returned once. Comparing this method with the simple loop approach showed that the reshape-sum method took on average four times fewer cycles to bin an array.

3.2.4 .FITS Cubes

Data is saved in .csv files if specified. Images are saved as .FITS files including headers[8]. For multiple runs, images can also be save as .FITS cubes. All generated frames are saved in standard .FITS format with the same header. The name of the generated file is:

```
frames/SXI_[block/single]_[callib/science]_[full/local noise]
_[A/B/A+B]_[Nr of frames]frames.fit
```

The header information can be found in Tab. 3.4 and is created with the functions `write_hdul` and `save_fits`, both utilizing the `astropy.io - fits` library, and can be found in the utilities file.

[8]Note, they can also be saved as .png files.

Information	Explanation
DATE-OBS	YYYY-MM-DDT hh:mm:ss observation start, UT
EXPTIME	Exposure Time in Seconds
SETTEMP	CCD Temperature Setpoint in C
CCDTEMP	CCD Temperature at start of Exposure Time in C
PIXDIM	Native Pixel SIze in microns
XPIXSZ	Pixel Width in microns (after binning)
YPIXSZ	Pixel Height in microns (after binning)
XBINNING	Binning factor in width
YBINNING	Binning factor in height
READOUTM	Readout Mode of Image
IMGTYPE	Type of Image
NOISE	Type of Noise
IMGFRAME	A, B, or A+B
FOCALLEN	Focal length of telescope in cm
INSTR	Soft-Xray Imager Data Sim for SMILE Mission
OBSERVER	Observer
NOTES	Additional Notes

Table 3.4: *Standard DaSi .FITS header information.*

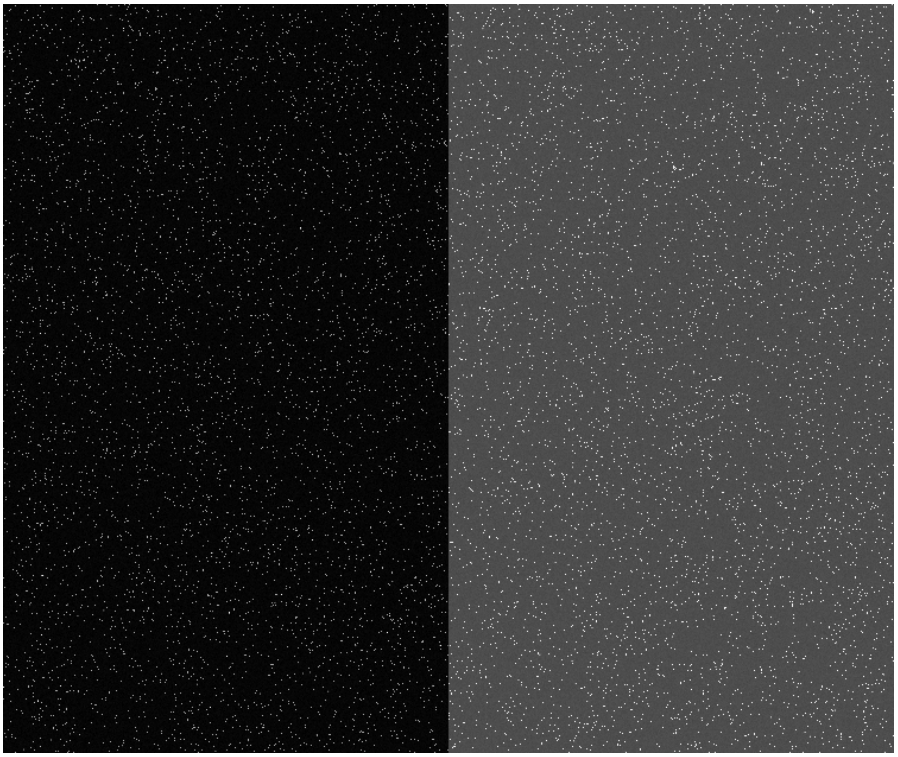

Figure 3.19: *An example of a science frame of 6000 eV events created by The Open University. It represents a sample of a CCD CCD370 frame, set to 6x6 binning split over two nodes.*

3.3 Comparisons

3.3.1 Open University - CCD370

Values and distributions were implemented rather empirically based off of images released by OU. Examples were given of images taken of 1 keV and 6 000 eV events (an example of one of these images can be seen in Fig. 3.19). This image includes the typical noise of a detector, however, it does not include effects such as a flat field, nor cosmics as they are to be expected of extraterrestrial observations.

A comparison of the histograms of the noise in OU image and a DaSi frame (one CCD) can be seen in Fig. 3.20.

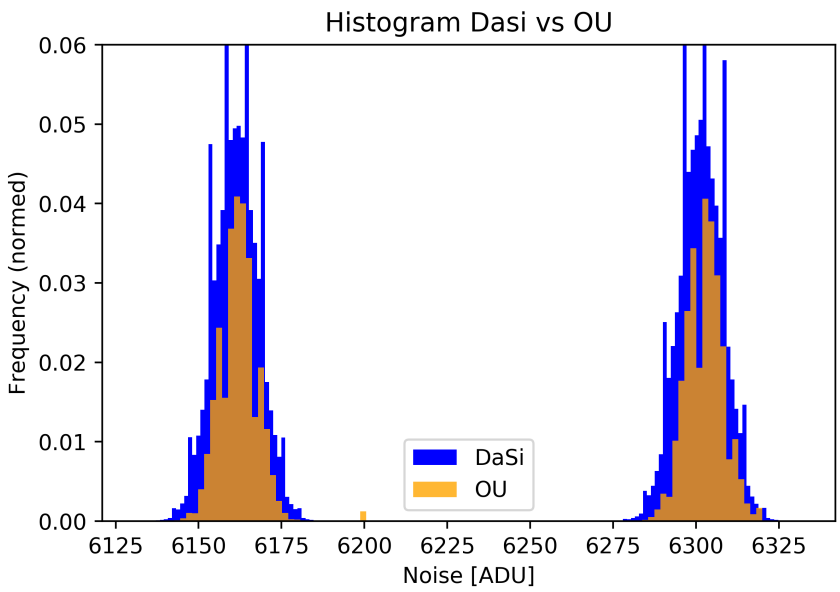

Figure 3.20: *Histograms of Dasi (blue) and OU (orange) noise.*

3.3.2 StarSim - uVie

StarSim is a previous detector simulator capable of modeling stellar fields as well as recreating instrumental features. It implements the previously mentioned PSFs, sub-pixel sensitivity as well as further glitches such as hot pixels (Ferstl, 2016). An example output of *StarSim* can be seen in Fig. 3.21. As is the case with the other simulators, this algorithm does not meet the requirements of running in real time. However, certain methods from this code (such as the PSF implementation) are planned to be adapted to DaSi (see Sec. 4.2).

3.3.3 ESA / ESO - Pyxel

Pyxel[9] is an open source end-to-end framework for detector simulations in development by ESA in cooperation with ESO. It will be able to include effects ranging from detector quantum efficiency, running temperature, Charge Transfer Inefficiency (CTI), Point Spread Function (PSF), readout nodes, pixel thickness, and more (Lucsanyi et al., 2018). An example of an output can be seen in Fig. 3.22. However, this pipeline only

[9] https://esa.gitlab.io/pyxel/doc/overview.html

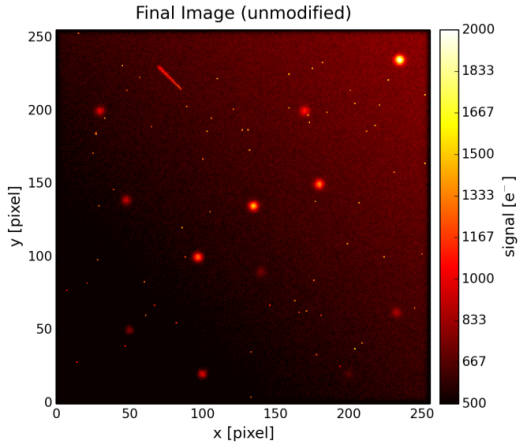

Figure 3.21: *An example of an image generated by StarSim (Ferstl, 2016).*

Figure 3.22: *An example of detector effects added to a user provided image created by Pyxel Lucsanyi et al. (2018).* ***Left:*** *Original image from the Hubble Space Telescope.* ***Center:*** *Added cosmic rays.* ***Right:*** *The addition of CTI and a lower full-well.*

adds these effects to an image, or images, uploaded by the user. A beta release is not public yet, but a request has been made to ESA if receiving a test-version would be possible. This simulator is mentioned for completeness, so as to further expand the knowledge of the forms in which simulations can be done. It could also be used as a method of adapting provided images like those from OU, and comparing them to results generated by DaSi.

3.4 Future additions

The current version of DaSi provides usable benchmark data and is currently being implemented at the institute for software testing. New releases are however already being worked on, meaning that new features will be added over time. Due to the modular style that the simulator is written in, adding additional functions is simple.

One such planned addition is the ability to add a PSF over the image. A point spread functions describes the response (blurring) of an image to a point source. This was not implemented in the original simulation since, due to the large binning, the relative small size of events, and the design of the payload, these effects are negligible. However, if applied to other missions, this may not be the case. The general form would be to add a function into the core library that reads a PSF file and convolves the science frame before binning.

Another addition which is still in beta mode, is the ability to account for different scenarios (for example: sun facing or high cosmic rate). However, due to the sheer amount of possible scenarios, and the high dependency on the type of mission, this will be a feature that needs to be tweaked for each new implementation. More to this in Sec. 4.1.

The inclusion of a bad pixel map is also ongoing, but this is a process that includes collaboration with the development of the rest of the data chain, since the DPU will include a bad pixel map to avoid passing events that land on, or near, these defective pixels.

Also, more detector effects are planned to be implemented, such as CTI, varying quantum efficiency, and the ability for an ambiguous amount of detectors (currently only up to two are possible). However, these additions will certainly reduce the speed of the simulator, and need to be precisely tailored to the type of mission that is to be implemented.

Chapter 4
Application

It's still magic even if you know how it's done.

Terry Pratchett

4.1 SMILE Data sets

The default software delivers benchmark data sets comparable to the OU frames. Special cases have been developed to test the SXI software and hardware. Automating the special cases, instead of being created ahead of time, is still ongoing. An extra configuration parameter will define which scenario is run. Current scenarios are:

CASE 0: Single image / Normal

CASE 1: Worst case -> max amount of expected X-ray events (see Sec. 3.2.2)

CASE 2: Sun facing (a. increasing and b. sudden)

CASE 3: Background pulses

CASE 4: Detector malfunction

CASE 5: Temporary influx of cosmics

CASE 6: Short bursts

Currently, these cases can be created by going into the config .xml file and changing the amount and intensity of both the cosmics and events and running the simulator. Once standard cases are implemented, the parameter can be set and a standard amount of runs will be executed, creating a .FITS cube of all images. Whether or not this option can be called directly by the FEE (for testing cases) is To Be Determined (TBD). Images of the sun-facing and cosmics cases can be seen in Fig. 4.1. A plot of a few of these cases (Normal, Sun facing, cosmic influx, tumbling satellite, and a short bright burst) with their total signal along multiple frames can be seen in Fig. 4.2. The cases are meant to check that the software reacts as it is supposed to, for example, if highly illuminated frames are captured for more than three frames[1], the radiation shutter door should be triggered and closed to avoid potential damage to the detector.

Fig. 4.3 shows snapshot histograms (*top*) and surface profiles (*bottom*) of one sun-facing case.

4.2 ARIEL

One further implementation which already uses DaSi is the ARIEL mission, for which the Department of Astrophysics is also tasked with developing data processing software. ARIEL aims to answer questions oriented around the formation of exoplanets and their atmospheres. To observe atmospheric composition and thermal structure, ARIEL will observe planetary transits, direct emission/transmission of planetary eclipses, phase curves (star+planet flux), and time series of narrow-spectral band observations. ARIEL has two spectrometers on board, the AIRS (ARIEL InfraRed Spectrometer) and NIRspec (Near-InfraRed Spectrometer). These are also planned to be simulated with DaSi. At the moment, no released application for ARIEL exists, but prototypes based off of DaSi can recreate existing images of both detectors. Example images from AIRS and NIRspec can be seen in Figs. 4.4 and 4.6. One main challenge for this adaptation will be implementing PSFs. Only in this case, since PSFs are wavelength-dependent, the PSF will need to be able to adapt along the spectrum. A preliminary example of the DaSi simulation for ARIEL AIRS and NIRspec are depicted in Figs. 4.5 and 4.7.

[1] This is defined as 50 % of all pixels above a certain threshold (currently set at 10 000 e- per pixel.)

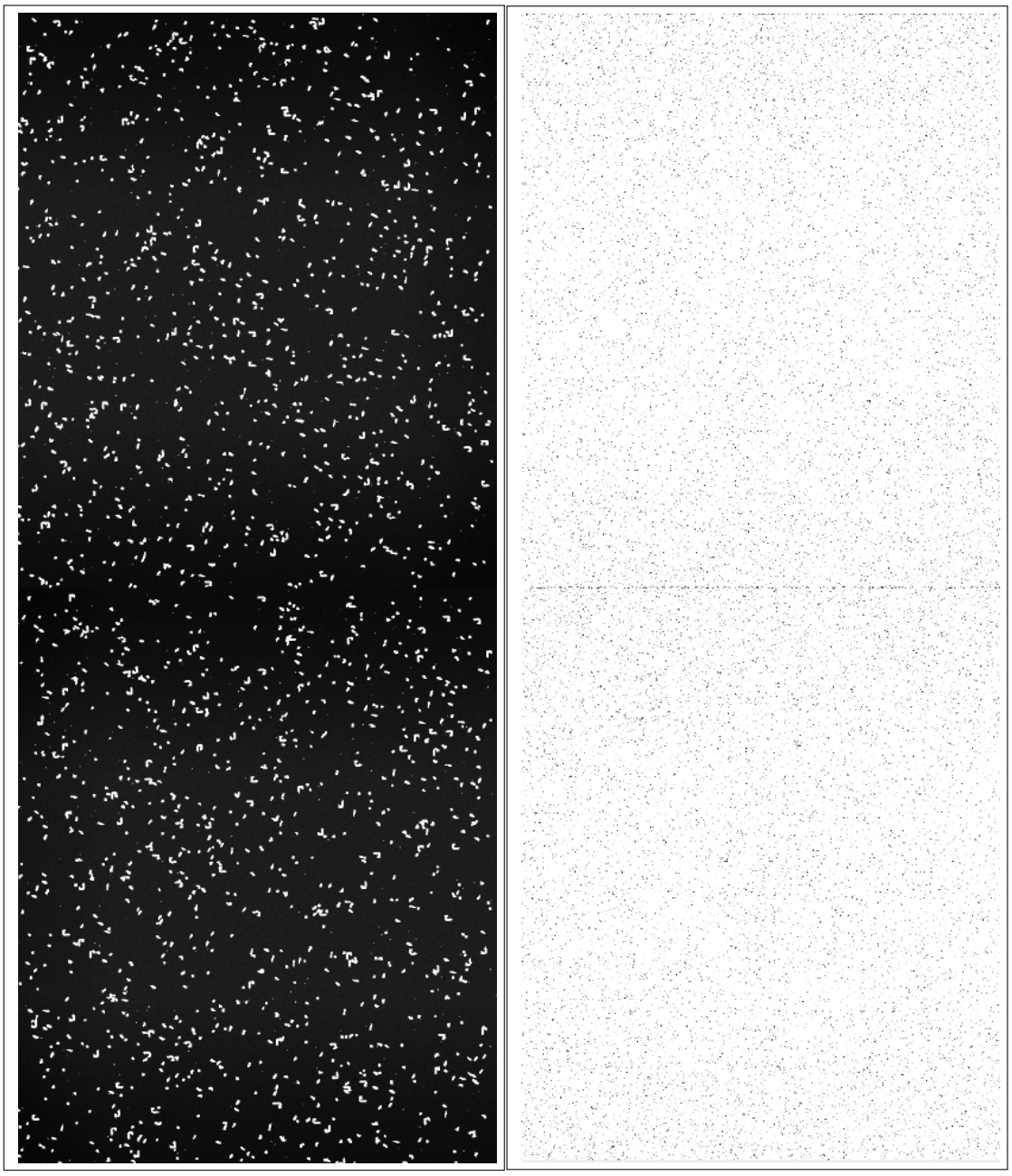

Figure 4.1: ***Left:*** *Cosmic influx case.* ***Right:*** *Sun facing case at worst setting.*

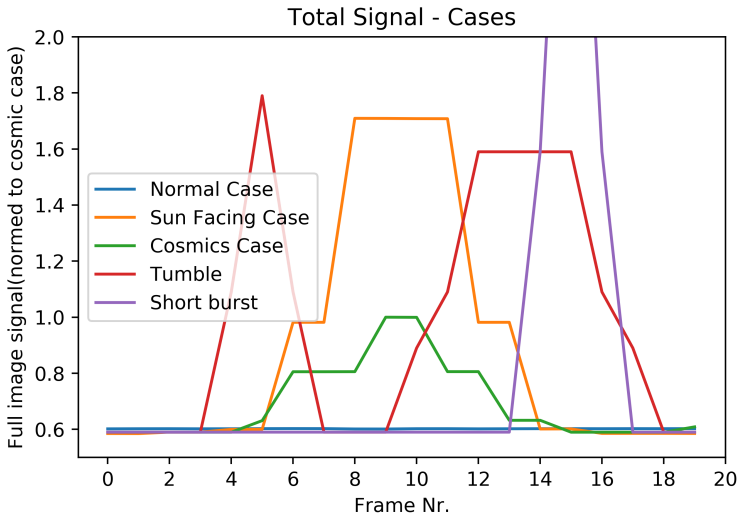

Figure 4.2: *Comparison of different cases created with DaSi. The normal case has a very small variation since the amount of signals and their intensity are always relatively similar. The sun facing case assumes the satellite's alignment is off and sun light enters past the straylight shield. The cosmic case is for the eventuality that a sudden burst of high energy particles enters the detector. If this happens for more than three images in a row, the FEE will command the radiation shutter doors to close until communications with earth is taken up on the next fly-by. The tumbling case assumes that the satellite has started to tumble, and erratically points towards the sun. The short burst case assumes two frames of very high energy, in this case, the radiation shutter should not be triggered.*

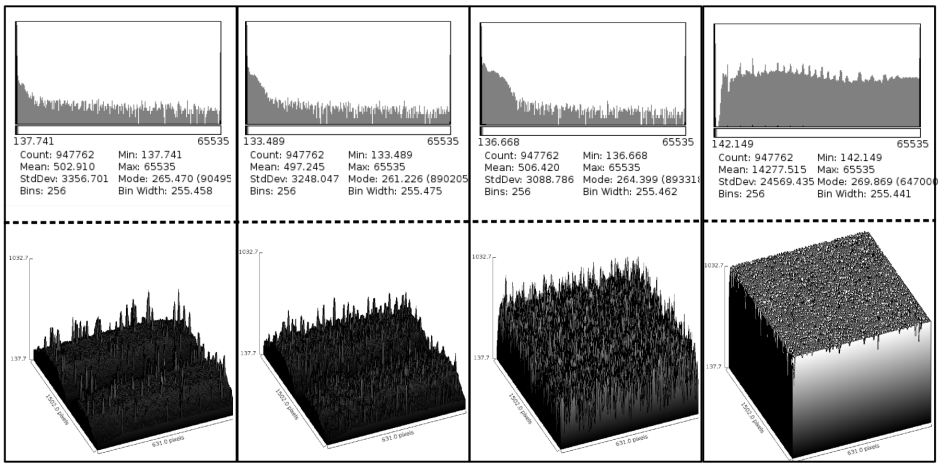

Figure 4.3: *A look into a sun facing scenario. The signal increases until the entire frame is nearly saturated.* ***Top:*** *Histograms, and* ***Bottom:*** *surface profiles.*

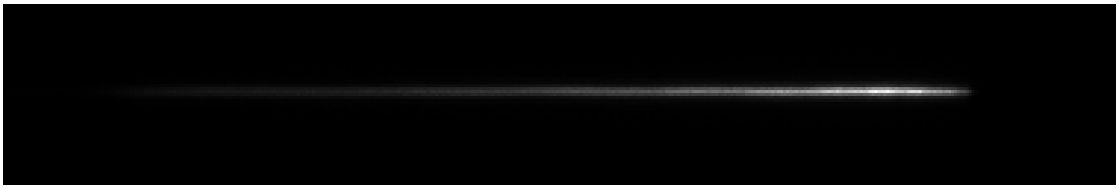

Figure 4.4: *Output from ARIEL's AIRS.*

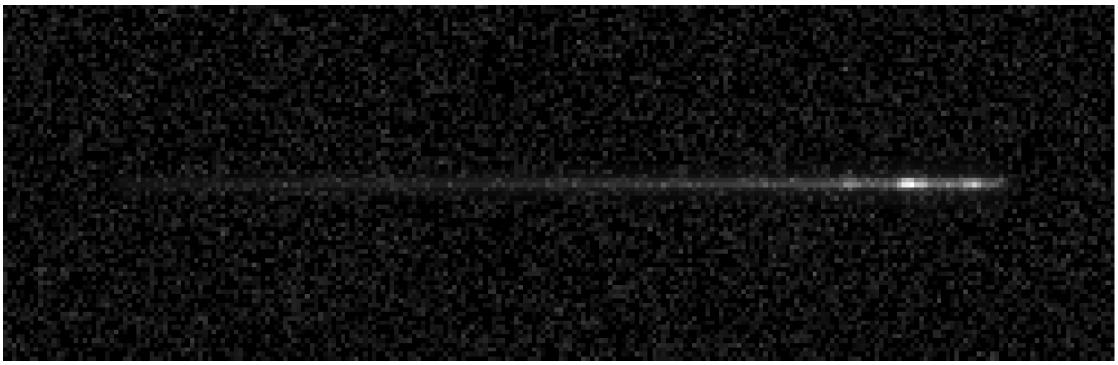

Figure 4.5: *Preliminary output of DaSi applied to ARIEL's AIRS. AIRS has a spectral range from 1.95 to 7.80 µm over two channels, meaning that the PSF varies enough to have an effect. This method of adapting the PSF is still TBD.*

A varying jitter (caused by the movement of the spacecraft) will also be relevant in these simulations. At the moment, a spectrum and certain lines are placed in the simulation itself, but it is planned to import a file of lines in a modular method similar to how the flat field is implemented in DaSi.

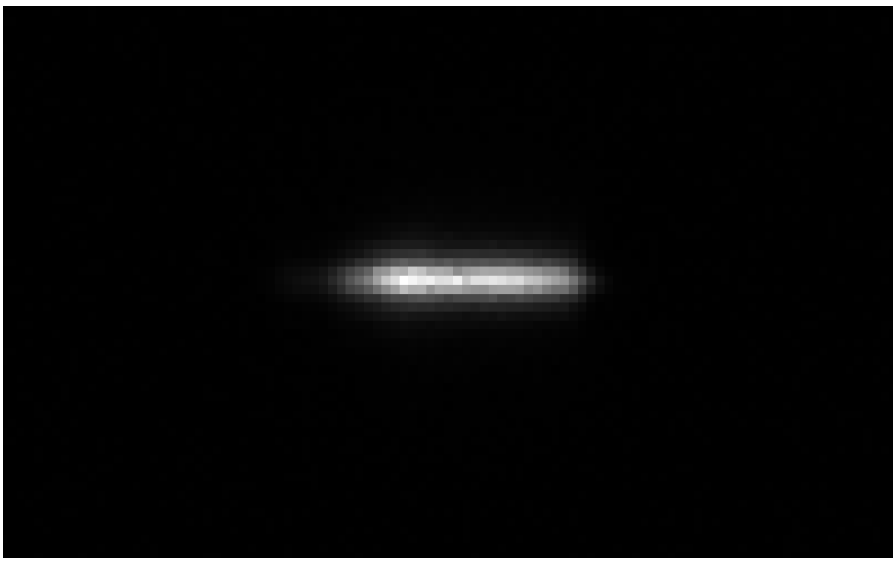

Figure 4.6: *Output from ARIEL's NIRspec.*

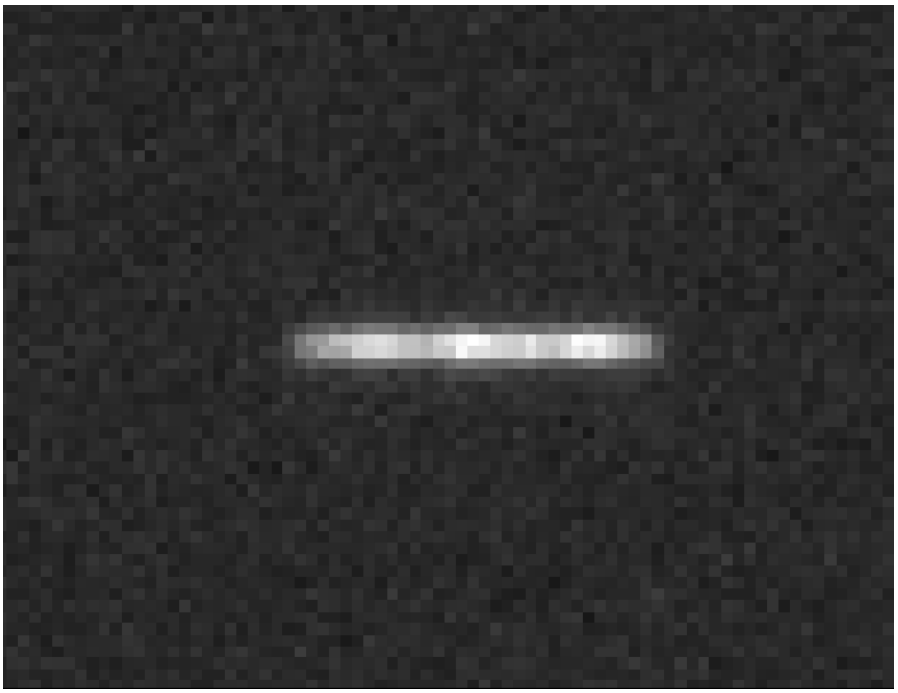

Figure 4.7: *Preliminary output of DaSi applied to ARIEL's NIRspec. This image is currently convolved with one PSF, namely for the central wavelength, since NIRspec has a very small wavelength range from 1.10 - 1.95 µm the PSF does not change significantly.*

Chapter 5
Data Processing

"Go back?" he thought. "No good at all! Go sideways? Impossible! Go forward? Only thing to do! On we go!".

Bilbo Baggins

Now that the data exist, it can finally be used to test the rest of the data chain. This process includes: detecting events from the exposed images; determining whether or not those detected events are in fact SWCX X-rays or non X-ray events such as cosmics or hot pixels; compressing and packaging the data; as well as storing it for transfer to Earth.

5.1 Data Chain

The Central Checkout System (CCS), which is used to send and receive packets as well as run test-scripts, will be connected directly connected to the On Board Computer OBC via the Packet Utilization Standard (PUS) with TCP/IP connection (communication protocols). The exact protocols between other components are TBC, with the space segment consisting of SpaceWire routers and most likely RMAP. The simulator chain can be seen in Fig. 5.1.

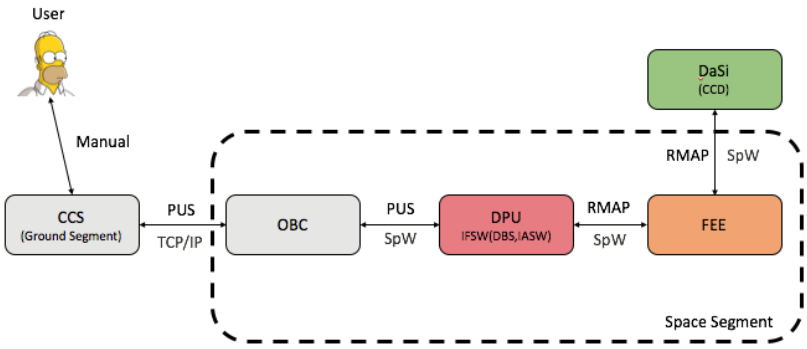

Figure 5.1: *Full SXI simulator chain.*

5.1.1 OBC

The On-Board-Computer is first and foremost applied for data storage between observation phases and data-transfer stages (Earth fly-by). Currently, the OBC simulator serves only as a connection link for the DPU and to check for configuration parameters.

5.1.2 DPU

The Data-Processing Unit is the main port between the SXI analyzer and spacecraft. It is split into two main parts, the DPU Boot Software (DBS) and Instrument Application Software (IASW). It will be the main source of event-sorting and data compression (approach for compression is TBD) and functions as the central component between the SXI analyzer (FEE and Radiation Shutter Door (RSD)) and spacecraft. This includes sending commands to the FEE (such as observation parameters and shutter functions), as well as returning Event data and housekeeping (HK)[1] to the OBC. The DPU receives a string of pre-processed 5×5 pixel events from the FEE and further processes these events (more in Sec. 5.2). Communication between the FEE and DPU will be over a SpaceWire connection allowing for transfer rates of 37.5 Mbit/s to 75 Mbit/s (double-edge clocking) and is therefore not a limiting factor. Data transfer between the DPU and PLM will also be with a SpaceWire.

[1] Data relevant to the maintenance of the satellite, such as temperature, heartbeat, storage capacities, and power.

It will have maps of misbehaving pixels. These are tables that can be updated from ground stations during the comms portion of orbit. They include: column/row offsets, global-node thresholds, pixel defect locations, and temperature dependent corrections. Pixels or columns damaged by radiaion or defects are determined in calibration runs and are stored in the DPU as filter-masks for incoming event packets from the FEE to avoid false X-ray detection.

The DPU will also be responsible for data volume management. The DPU is also in control of monitoring the spacecrafts storage space. Several data storage modes that are TBD will be implemented depending on the fraction of storage volume remaining and observational time left before data transfer to earth. The basic building blocks of the SXI DPU (Jeszenszky, 2019) are:

- LEON3 processor for data processing and communication with the P/M
- FPGA for ADC control and communication with PSU and RSE
- ADC for housekeeping data acquisition from the camera
- 16 channel multiplexer for gathering PSU HK and temperatures
- Current source for controlling the CCD and FEE thermistors
- PoL converter for generating the required +1.5 V and +1.8 V board voltages
- PROM holding the boot software
- EEPROM holding the application software
- SRAM for program execution and data storage

The DPU memory map can be seen in Fig. 5.2.

5.1.3 IASW

The IASW is one sub-part of the DPU and provides services to SMILE's SXI Platform Module (PLM) ICU and is a user of the Front End Electronics (FEE). It is comprised of:

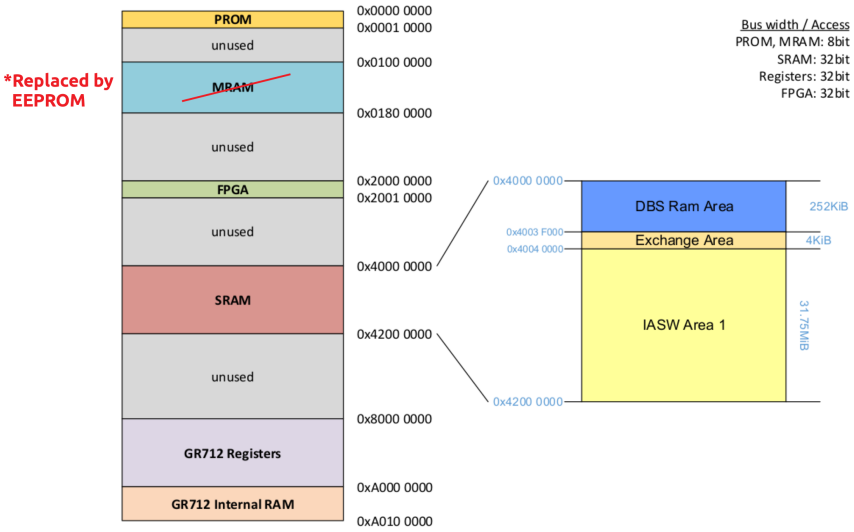

Figure 5.2: *DPU Memory Structure: overall and Ram (Jeszenszky, 2019).*

- A *boot software* (provided by IWF, who are building the hardware)
- A *basic software* (BSW) which is the operating system, communication links to the FEE and PLM ICU, and DPU hardware management functions
- The *application software* (ASW) which provides high-level services (Reimers et al., 2019)

This structure can be seen in Fig. 5.3. The communication between PLM ICU / IASW / FEE is built on SpaceWire (SpW) with Remote Memory Acess Protocol (RMAP) based packets (ECSS-E-ST-50-12C, 2008) to the FEE and Packet Utilization Standard (PUS) based packets (ECSS-E-ST-70-41C, 2016) to the PLM ICU and Ground.

The DPU hast four main states: *Standby*, *FEE_Offline*, *Pre_Science*, and *Science*. These states are depicted in Fig. 5.4.

5.1.4 FEE

The FEE simulator connects directly to the CCD / DaSi and preprocesses the incoming data into event packets. The FEE hardware is baselined as a single Field Programmable Gate Arrays (FPGA) (BRAVE) for all four CCD outputs (2 CCDs each with 2 nodes).

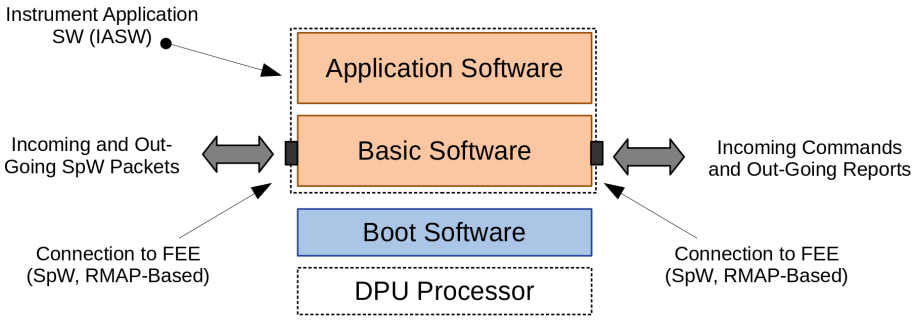

Figure 5.3: *Structure of SXI Instrument Application Software (Reimers et al., 2019).*

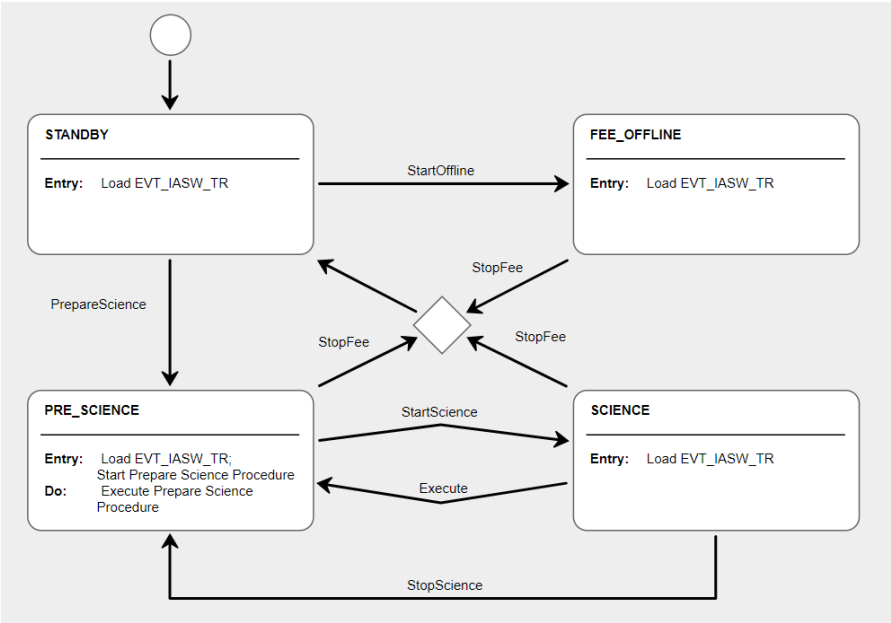

Figure 5.4: *DPU State Machine.*

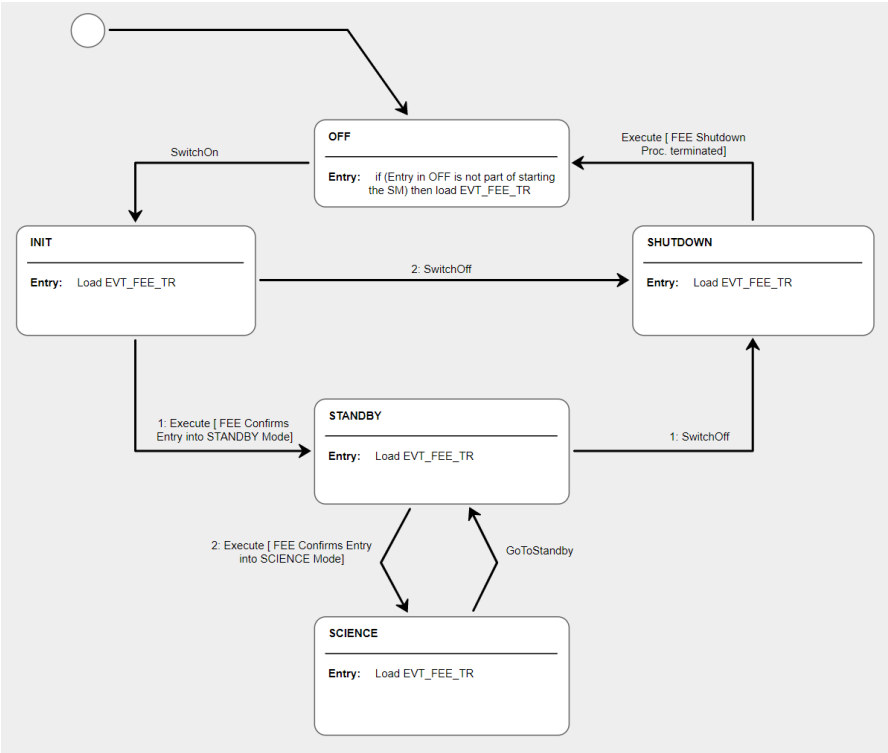

Figure 5.5: *FEE State Machine.*

The included SRAM (Static Random Access Memory) retains its data bits in memory until powered off and does not need to be refreshed like a standard dynamic RAM. Using half the built-in SRAM allows for 5 rows to be held at a time (16 bit/pixel with a row of 385 binned pixels per node). This means that event detection will have to take place in real time as the CCD is read out. The bitrate will be 37.5 Mbit/s (75 Mbit/s for double-edge clocking) and is therefore not a limiting factor.

The FEE has five states, starting in *Off* mode, then, moving to *Init*. From there it goes into *Standby* where it waits for a command to go into *Science* mode, and when needed, the *Shutdown* state. The FEE state machine is depicted in Fig. 5.5.

Event Data Packet:

Each event data packet breakdown is conceptualized to keep the most useful information but to minimize the data per event, and can be seen in Tab. 5.1.

Information	Description	Bit Width	Values
CCD#	Which CCD event was observed in	1	0 = CCD1, 1 = CCD2
Column#	Which px column was center	10	≈770 px per CCD row (Max 1024)
Row#	Which px row was center	10	719 rows in store region (Max 1024)
Pixel Data	Raw pixel data of 5×5 area	300	12 bit × 25 px

Table 5.1: *Breakdown of event data packet*

FEE - DPU communication:

A flow chart of data processing between the FEE and DPU can be seen in Fig. 5.6. The FEE data packets will also include a flag reserved for sun-facing security measures. If 50% or more of the pixels are above a certain threshold (currently set at 10 000 e- per pixel), the flag is activated and will tell the DPU to initiate the radiation shutters to close. The DPU and FEE state transitions are presented in Tab. 5.2.

5.1.5 CCD

The CCD is planned to allow for integration times between 0 to 100 seconds with a resolution of 0.01 s (TBC) with a potential implementation of a Transistor-Transistor Logic (TTL) clock.

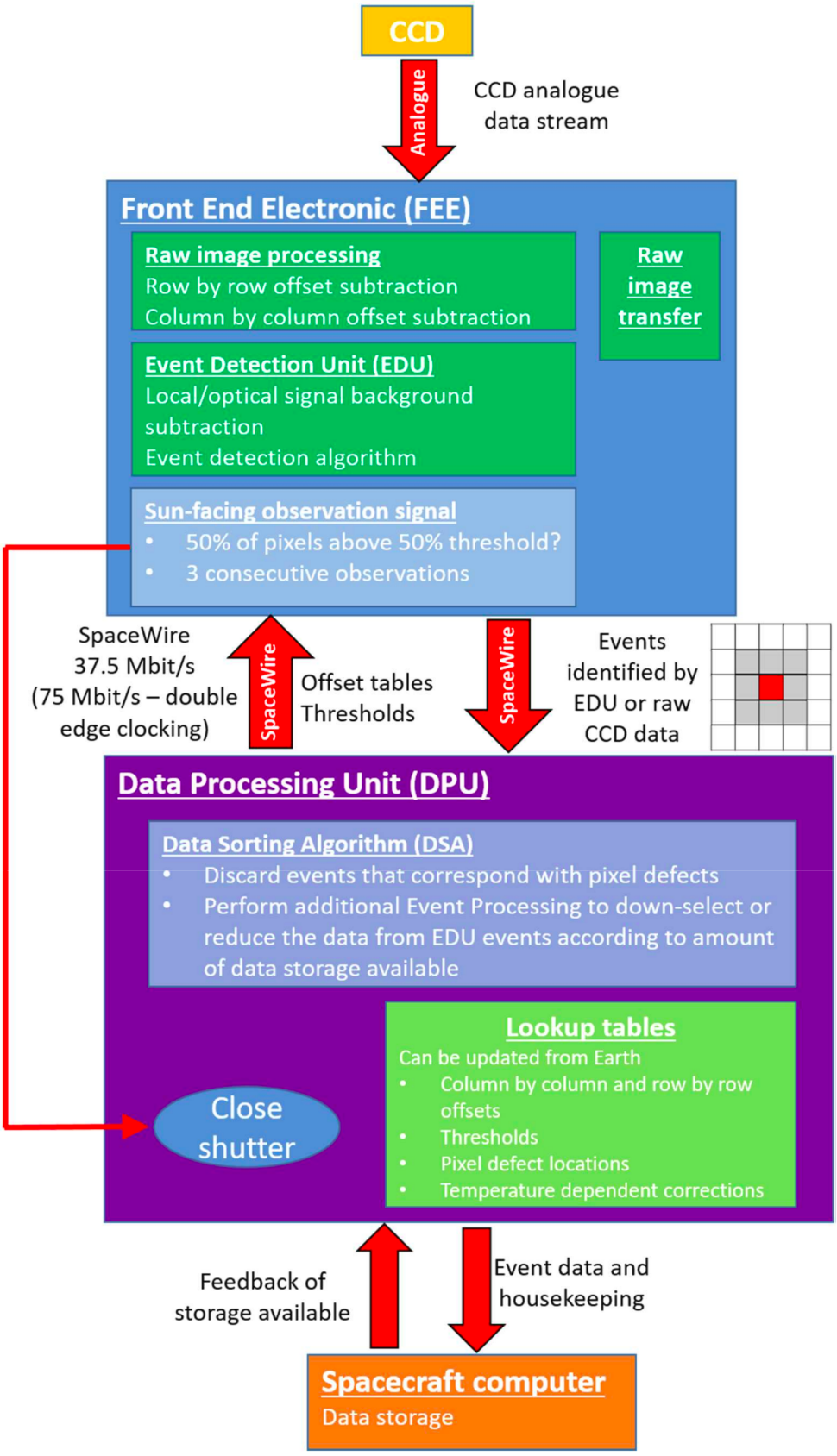

Figure 5.6: *Flow chart of data processing in FEE and DPU.*

DPU SM	FEE SM	DPU	FEE
		unpowered	unpowered
		powered, booting	unpowered
STANDBY	OFF	DPU loaded and running	unpowered
	INIT	SwitchOn Procedure	powered, power-up sequence
	STANDBY		confirms ON mode, HK available
PRE_SCIENCE	STANDBY	Prepare Science Procedure	configuration written in register map
SCIENCE	SCIENCE	Get Science Data	Exposure starts
STANDBY	STANDBY	StopFEE	execute op set to false
STANDBY	SHUTDOWN	SwitchOff Procedure	ccd_mode_config to standby
STANDBY	OFF		unpowered

Table 5.2: *State Machine Transitions*

Operation modes of the CCD can be seen in Tab. 5.3. These include:

- Binning modes
  a. 1×1 - Diagnostics and Calibration
  b. 6×6 - Event detection
  c. 24×24 - Secondary science mode (UV detection)
- Idle Mode - Only DPU active
- Safe Mode
- Off

5.1.6 CCS

The Central Checkout System (CCS), developed at the institute for testing of CHEOPS software, is used to run test scripts which control the instrument through Telemetry Commands (TC) and collects the Telemetry Packets (TM) of various kinds. It can be seen as the "Ground Station" with enhanced capability to also allow a quick look at the instrument response in real-time, such as house-keeping (HK) inspection, but also the display of science data in near real-time. It is available as a virtual machine for download and runs on any up-to-date Linux distribution. It has a graphical user interface (implemented utilizing GTK 3) and function library (used for communication with the instrument). See Fig. 5.7.

The CCS editor is started by running `ccs.py` in the root directory. From there, additional libraries can be included. A TMTC (telemetry packet/command) pool can be created by connecting to the respective TM/TC sockets (for sending and receiving packets). A more detailed explanation of getting started with the CCS can be found in CHEOPS IFSW: CCS User Manual Sec. 9.3. A plotting window and data pool filtering feature are also included.

CCD Mode	Characteristics	Readout	Usafe	Action
6 binning	6×6 binning, event detection	Event detection	Science Mode, Calibration	Detecting soft X-rays
24 binning	24×24 binning	Frame transfer, windowed	Secondary Science Mode	Detecting UV photons
No binning frame transfer	-	Frame transfer, windowed	Diagnostics and Calibration Mode	Failure analysis mode, In-orbit, Ground calibration data
No binning full transfer	-	Frame transfer readout, windowed	Diagnostics and Calibration Mode	Failure analysis mode, In-orbit, Ground calibration data
Idle Mode	-	-	DPU on only, no active CCD readout, RSM closed	Passage through -radiation belts and perigee
Safe Mode	-	-	DPU on only, no active CCD readout	Instrument safe
Off	-	-	Instrument completely switched off, S/C survival heaters on	-

Table 5.3: *CCD Operation Modes*

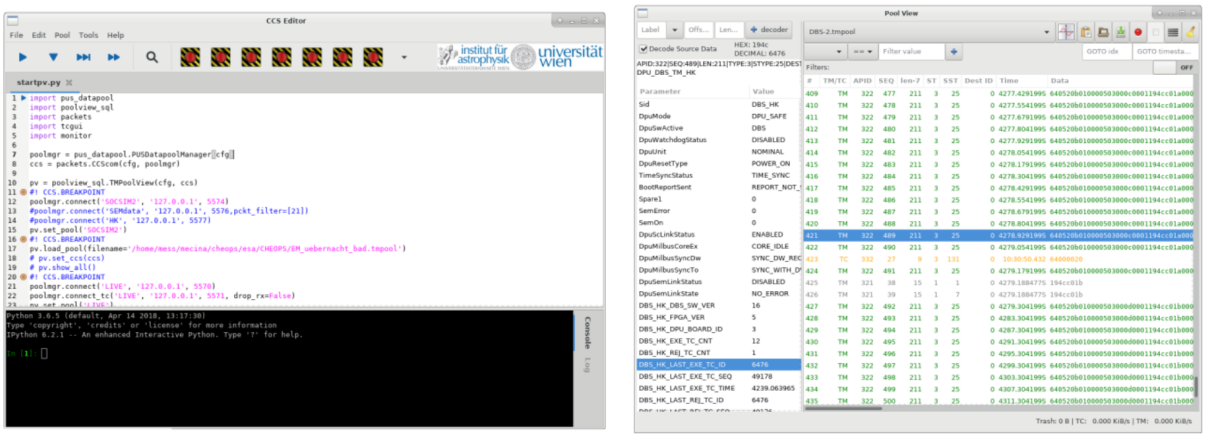

Figure 5.7: ***Left:*** *A loaded test script in the CCS opening screen.* ***Right****: The CCS pool viewer with a selected HK packet, which is decoded via the IDB in the left section.*

5.2 Detection Rules

5.2.1 FEE Data Sorting Algorithm

The FEE scans the incoming (CCD or DaSi) data and scans it for events. The Event Detection Unit (EDU) is applied on a pixel by pixel basis, inspecting pixels along a row using the full 5 rows in the FEE's SRAM in real-time. If the following criteria are flagged as true, the inspected pixel is identified as an "event":

- The pixel contains more signal than a specified "single pixel threshold" plus the local signal background.
- The pixel is a local maximum with more signal compared individually to its surrounding 8 nearest neighbors.

The single pixel threshold will be unique to each CCD output node, and stored in the DPU registry map, and will be defined as $\approx 4.5\sigma$ of the read noise (TBC). The local signal background is calculated as the mean of the outer most 11 (or 14) pixels. The top three pixels in the farthest corner from readout direction may be neglected in this calculation due to possible loss of charge during transfer (charge transfer inefficiency - CTI), see Fig. 5.8.

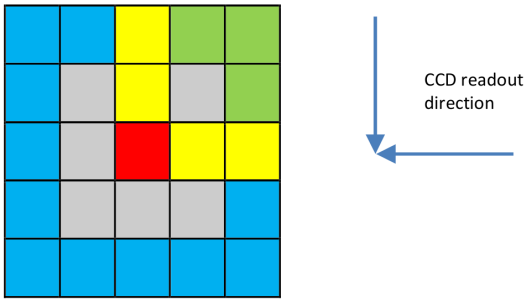

Figure 5.8: *Schematic of pixels around an event.* ***Red:*** *central pixel with highest signal,* ***Blue:*** *used to determine local background,* ***Yellow:*** *pixels which may include CTI,* ***Green:*** *pixels which may be subject to second order CTI.*

5.2.2 DPU Data Sorting Algorithm

The 5x5 px areas sent from the FEE will be interrogated for storing or rejection as events as follows (TBC):

- If the center pixle has too large a signal, it will be classified as a non-X-ray event.
- If the sum of the 3×3 px area surrounding the main pixel has too large a signal, it may be classified as a non-X-ray-event.
- The number of pixels in the 3×3 area that have a greater threshold pixel value can be counted in order to classify the event (Pattern recognition, see Fig. 5.9).

These events may be cosmics, may be classified as lower priority data, and may not be fully transmitted to the ground.

5.3 Simulator Operation

The entire data chain for SMILE is available in the OBSW (On-Board Software) git repository as released by uVie IfA. The simulators can be built via a `makefile` by simply running the command `make` from a terminal in the parent directory where the `makefile` resides (this includes installing all dependencies: configinator, testlib, databases, and setting permissions). Alternatively, just compiling the needed files can be done by running the command `make build-pc`. Individual portions of the simulator can

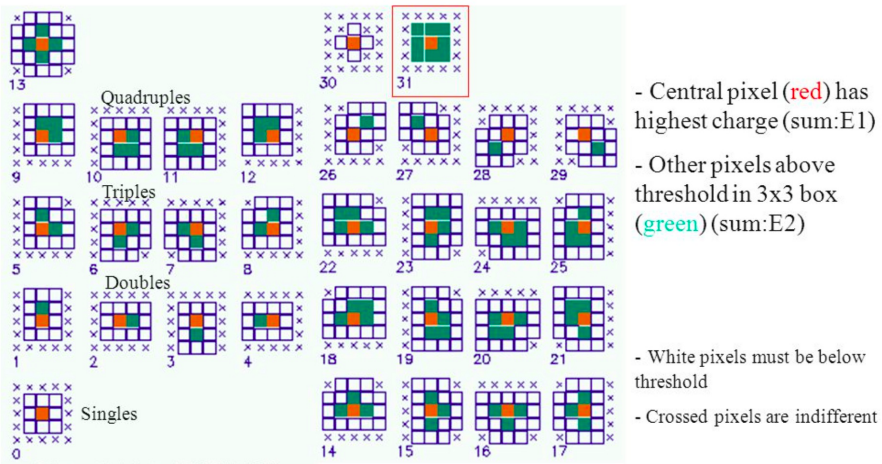

Figure 5.9: *Expected forms of SXI events in 6x6 binned format.*

be compiled by running `make build-xxx` with `xxx` being the needed portion (e.g `fw-profile`, `crplm` (PLM), `cria` (IASW), and `crfee` (FEE)).

There are currently three methods of running the SMILE data processing chain once compiled. Either (1) via terminals, (2) with a user interface developed by the institute in Vienna, or (3) via a Raspberry-Pi module developed by the institute in Graz.

(1) Terminals:

The bash file `open_components` opens terminals in the respective build-directories for the PLM, IASW, FEE, and DaSi. From there, the individual components can be started by executing their bash files. These commands can be found in Tab. 5.4. Note, that the DaSi is started via a socket-server that connects the FEE to the data simulator.

An example of the layout of all terminals running their respective components can be seen in Fig. 5.10. Each termial can be started individually and each component will open a socket to connect to their respective partners and wait for a response, meaning that the order in which they are started is irrelevant. It is important to note at this point, that as soon as a connection is established, the simulators will not cyclically check the connection. This means that, if one of the segments is interrupted, the remaining segments will keep running (except for the socket between the FEE and DaSi).

Component	Command
PLM	`./CrPlm`
IASW	`./CrIa`
FEE	`./CrFee`
DaSi	`python socket_server_datasim.py`

Table 5.4: *Data Processing Simulator Commands*

(2) User Interface:

A Graphical User Interface (GUI) has also been created with adaptability for other projects in mind (Fig. 5.11, created by Marko Mecina). It is built in such a way that, any needed "building blocks" can be called (in the case of SMILE, this refers to the PLM, IASW, FEE, and DaSi socket). These building blocks can then be connected in any given constellation by their resepctive socket points. Each block can be started individually and the respective terminals can be blended in if need be. A future version will also allow for the selection of the type of connection (for example: RMAP or PUS). This UI is however, still in development and has not yet been released.

(3) Raspbery-Pi Module

The Space Research Institute (Institut fuer Weltraumforschung (IWF)) in Graz has developed a Raspberry-Pi module with specs as they will be in SMILE to test the software on the appropriate hardware. The module can be seen in Fig. 5.12. It will run the DPU software, meaning it hold both the DBS and IASW software, and will communicate with the OBC (or Ground/CCS if specified) and the FEE.

The Raspberry-Pi module currently has two executables on it. The first (`DBSonRASP`) is for running the boot software (DBS), which is started as soon as the module is connected to a power source. The second executable (`IASW`) is the IASW software, which will start

Figure 5.10: *Example of terminal layout for uVie IfA's data-chain simulator.*

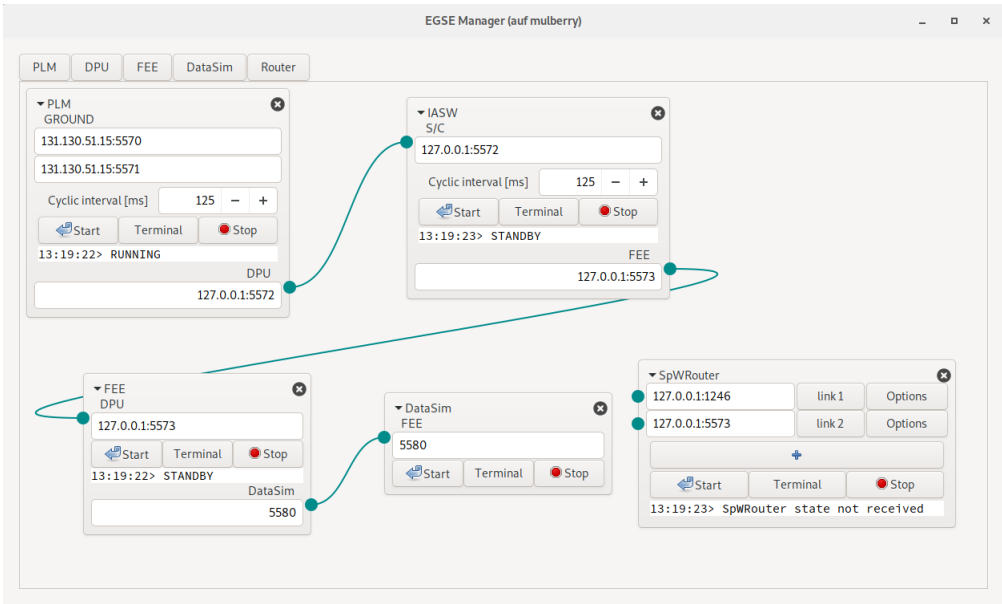

Figure 5.11: *Current version of the UI for the data-chain simulator. A top bar allows for the selection of "building blocks" which can then be connected via their respective sockets.*

when the appropriate telemetry command is sent by the OBC (`TC_BOOT_IASW`). Implementation of the FEE and DaSi simulators in the module is currently in development. The module can either be connected to (i.) directly over a network, or (ii.) over a PLM-SpaceWire Gateway.

i. Direct connection:
Since the module runs on Linux, it can be directly accessed, and the software updated, using a SSH connection with the following parameters:

- static IP: **192.168.0.1**
- username: **pi**
- password: **smile2020**

Once a LAN connection exists, the module can be accessed over the terminal with the command `ssh pi@192.168.0.1` and entering the password. The DBS and IASW software are located in the root file system of the simulator. The IASW software is started

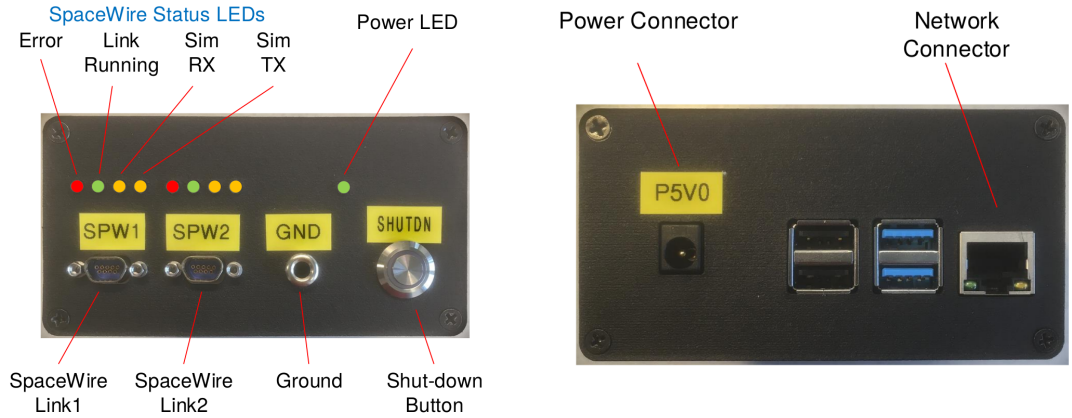

Figure 5.12: ***Left:*** *Graz DPU simulator front including SpaceWire links (SPW1 for connection to the spacecraft / ground, SPW2 for connection to the FEE), a ground which will be connected to the ground of the spacecraft and a shut-down button (SHUTDN).* ***Right:*** *Graz DPU simulator rear including a power connector (Module turns on directly when connected to power), and a network connector which can be used to communicate directly with the module.*

directly from the executable instead of from the nonvolatile memory or RAM area. More information about this can be found in Hasiba (2020). Note that this method is communicating directly with the raspberry-pi module, and is not the type of communication between Ground/OBC and the DPU (PUS packets via SpaceWire) that will occur in SMILE.

ii. PLM Gateway:

An Ethernet-to-SpaceWire gateway enables the routing of SpaceWire packets through an Ethernet interface. This can be connected to via the gate IP: `10.19.4.105` with the SpaceWire port 5000. This interface can be used to communicate to the DPU raspberry-pi module in the same manner as will be implemented in the spacecraft. The configuration interface for the gateway can be seen in Fig. 5.13. More information can be found in Martin-Serrano and Herrero (2019).

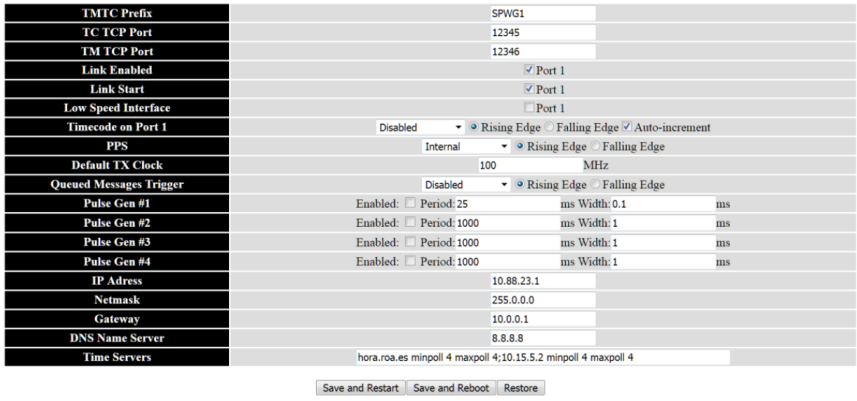

Figure 5.13: *Default configuration interface for the PLM SpaceWire Gateway.*

Code:

A few alterations were made to the simulator to adapt for the raspberry pi. After initial tests, DaSi still took on average 4.62 seconds. These alterations reduce the adaptability for other missions, but significantly increases speed time. They include changes such as loading the flat frame once in the socket server and passing it to DaSi, instead of being imported and scaled to the detector size each time DaSi is called. This eliminates the simple changing of flat fields on the fly, but ensures that the frame is only loaded into memory once. The cosmics are also all loaded once in a similar fashion. Resizing arrays, whether it be done by numpy, openCV, or any other method, has consistently taken up a large portion of the computational time. By taking an already scaled flat frame as well as precalculated and rotated cosmics, a lot of calculations can be spared. However, this also means that they cannot be changed as intuitively as before. Due to its speed, this adapted version was given the name RAPIDsim (RAspberry PI Data Simulator). The run times of the RAPIDsim can be seen in Fig. 5.14 for execution on the i7 processor, and in Fig. 5.15 for execution on the raspberry-pi module's 1.5 GHz Cortex-A72 processor. Very curious is the fact that the calibration run time on the raspberry-pi consistently decreases. This was confirmed over three tests of 100 runs each. This behavior does not show up in the i7 runs, nor does is occur when only the calibration is run as a stand alone program. It only appears when the function is called by RAPIDsim on the pi module. The source of this gradient is still being investigated. The average, minimum, and maximum run times for both versions can be seen in Tab. 5.5.

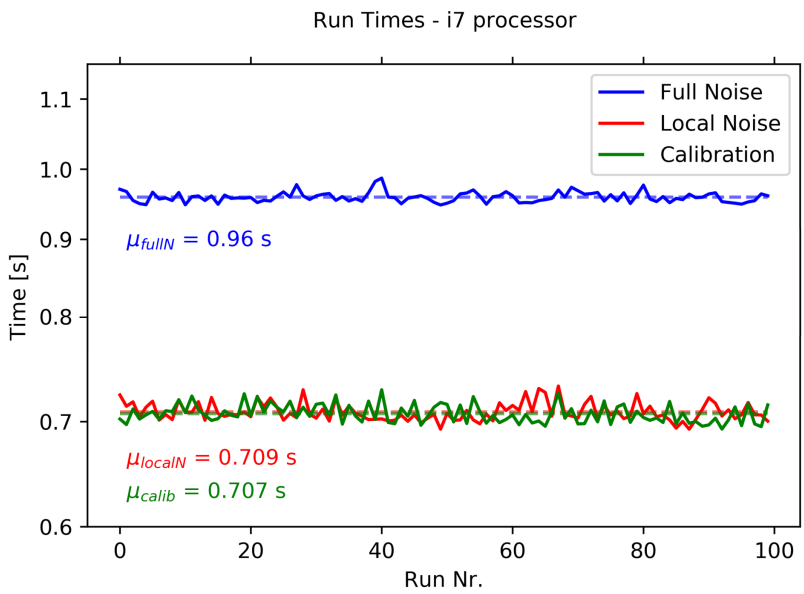

Figure 5.14: *Run time of the raspberry-pi configured DaSi for all three main cases running on an i7 processor.*

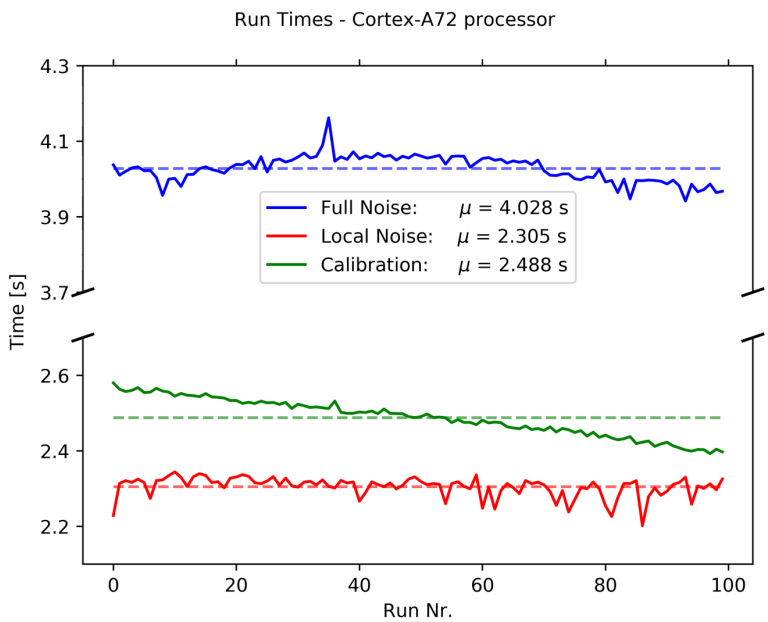

Figure 5.15: *Run time of the raspberry-pi configured DaSi for all three main cases running on the raspberry-pi 1.5 GHz Cortex-A72 processor. Noteworthy is the consistent decrease of the calibration run time.*

		Run Time [s]		
Processor	**Type**	**Minimum**	**Average**	**Maximum**
Intel-i7	*Full Noise*	0.955	0.961	1.005
	Local Noise	0.698	0.709	0.712
	Calibration	0.696	0.707	0.715
Cortex-A72	*Full Noise*	3.961	4.028	4.194
	Local Noise	**2.210**	**2.305**	**2.351**
	Calibration	2.378	2.488	2.596

Table 5.5: *RAPIDsim run times over 100 runs for each type and processor. Most important is the run time of the local noise case for the Cortex-A72 processor (Raspberry-Pi 4), and that it is consistently under four seconds.*

Interaction:

PLM:
Once started, the PLM will import configurations from `CrPlm.cfg` and establish socket points for a (hitherto undefined) ground segment, and the IASW, and then continue its cyclic activities: waiting for a connection with the IASW and consequently sending it the command to run.

IASW:
If the IASW is started alone it will read its config (`CrIa.cfg`) and initialize connections to the PLM and the FEE, and continually check for an answer from those ports. Once connections are established, the IASW can send commands to the FEE. Currently, it can set and get certain variables in the FEE. However, this library only consists of two variables at the moment: `set_vstart` and `set_toi_period`.

FEE:
The FEE similarly first established connections to the IASW and DaSi sockets and then, when commanded to by the IASW, sends an image request to the DaSi. It will also control the movement of the RSD. If the FEE receives an image from DaSi, it runs the data

through its established event detection procedure. In its end form, the FEE will mainly be in either *science* or *standby* mode.

DaSi:
The socket server for DaSi waits for a connection from the FEE. When it received a request, it unpacks the parameters, calls DaSi, serializes the resulting image, and transmits it back to the FEE.

5.4 TL;DR

A short recap of how to initialize and run the IfA SMILE data processing simulator.

Terminals:
This can be used as a quick-start to using the software via terminals:

i. Navigate to parent directory:
`OBSW/implementation/` (default)

ii. Compile

   a. Run makefile: `make`

   b. Alternatively, to simply compile each component, run:
   `./compile_all` or `make build-pc`
   *(Note, this **will not** verify and install potentially missing dependencies).*

iii. Open needed terminals by running
./open_components.

iv. Run each component respectively with:
`./CrPlm`, `./CrIa`, `./CrFee`, and `python socket_server_datasim.py`

Raspberry-Pi:
A quick-start to using the raspberry-pi module. For User Manuals on the DPU module and the PLM SpaceWire Gateway see Hasiba (2020) and Martin-Serrano and Herrero (2019).

i. The module can be directly accessed with

   a. `ssh pi@192.168.0.1` and entering the password `smile2020`.

   b. Running the IASW executable

ii. Or connect to the module through the SpaceWire Gateway

   a. Via the IP `10.19.4.105`.

   b. Sending TMs and TCs to the DPU with the implemented interface

Chapter 6
Conclusion

Don't cry because it's over. SMILE because it happened.

Dr. Seuss

This thesis has introduced the SMILE mission, its main science goals, and the data processing chain that the Department of Astrophysics of the University of Vienna is developing.

The goal of this thesis was to develop a simulator which could create a wide range of benchmark test data that can be implemented in a variety of missions, while also being effective enough to run in real time during the entire data-processing chain. The resulting DaSi simulator has been able to do just that. It is currently being implemented in development for two ESA missions (SMILE and ARIEL) at the institute. The simulator can be configured to create detectors of any given size, add parametrically generated noise, and events as defined by the configuration file. It can create specific scenarios to test both the software as well as the hardware for extreme cases. And it also has the ability to create preliminary spectra. The entire data processing can be run through emulators via terminals, with an integrated GUI, or partly over a hardware module (Raspberry-Pi).

In the case of SMILE, the data simulation DaSi can be run with either a full frame of noise, which can be run in real time on any processor of the caliber intel i5 8th gen. or

better, or run with only creating noise around the events and cosmics that are generated. The full noise frame method takes at least 2.52 seconds, on average 2.61 seconds, and up to 2.76 seconds. The local noise frames take on average 1.05 seconds to generate, with a minimum of 0.98 and a maximum of 1.24 seconds.

A simulation adapted for the rapsberry pi module (RAPIDsim) was created as well, reducing some of the adaptability/modularity, but significantly increasing the run time. It can create a local noise frame in at least 2.22 seconds, on average 2.31 seconds, and a maximum of 2.35 seconds; a calibration frame taking at least 2.38 seconds, on average 2.49 seconds, and a maximum of 2.60 seconds; and a full noise frame taking at least 3.96 seconds, on average 4.03 seconds, and a maximum of 4.19 seconds.

The simulator is constantly being updated and adapted for new cases and missions. It was programmed with this in mind, and was kept as modular as possible, with each step of the simulation being made easy to augment or remove, as well as including the addition of new functions without a dependence on other steps of the simulation.

I hope that DaSi can contribute further to the testing of software and that it can look forward to generating light for many missions to come.

Bibliography

Angelopoulos, V., McFadden, J. P., and Larson, D. (2008). Tail Reconnection Triggering Substorm Onset. *AAAS*, 321(5891).

Arentoft, T., Kjeldsen, H., Ridder, J. D., and Stello, D. (2003). The Eddington CCD Data Simulator. *Second Eddington Workshop: Stellar structure and habitable planet finding*, 1(59-64).

Boudouridis, A., Zesta, E., Zesta, E., Lyons, R., and et al. (2003). Effect of solar wind pressure pulses on the size and strength of the auroral oval. *Journal of Geophysical ResearchJ Atmospheres*, 108(8012).

Branduardi-Raymont, G., Agnolon, D., Li, J., and Colangeli, L. (2018). SMILE - Definition Study Report. *ESA*, 1(1).

Branduardi-Raymont, G., Sembay, S. F., Eastwood, J. P., and et al. (2011). AXIOM: Advanced X-ray Imaging Of the Magnetosphere. *Experimental Astronomy*, 403(443).

Bremer, J. and Nobels, F. (2015). Characterising the Cosmic Rays in a CCD. *Kapteyn Astronomical Institute - Basic Detection Techniques*, 1(1).

Burch, J. L., Moore, T. E., and Torbert, R. B. (2016). Magnetospheric Multiscale Overview and Science Objectives. *Space Science Reviews*, 199(1).

Clark, J. (2013). Rate constants and the Arrhenius equation.

Cravens, T. E. (2000). Heliospheric X-ray Emission Associated with Charge Transfer of the Solar Wind with Interstellar Neutrals. *The Astrophysical Journal*, 532(L153-L156).

Dennerl, K. (2010). High Resolution X-ray Spectroscopy of Comets. *Proceedings of the 'High resolution X-ray spectroscopy workshop', Mullard Space Science Laboratory*.

Driss, G., Addaim, A., and Abdessalam, A. M. (2018). Enhanced Box-Muller method for high quality Gaussian random number generation. *Int. J. Computing Science and Mathematics*, 9(287-296).

Eastwood, J. P., Hietala, H., Toth, G., and et al. (2015). What Controls the Structure and Dynamics of Earth's Magnetosphere? *Space Science Reviews*, 188(251-286).

ECSS-E-ST-50-12C (2008). ECSS-E-ST-50-12C - SpaceWire - Links, Nodes, Routers and Networks.

ECSS-E-ST-70-41C (2016). ECSS-E-ST-70-41C - Telemetry and Telecommand Packet Utilization.

ESA/ATG (2019). Unsteady magnetic reconnection in Earth's magnetosphere. Accessed: 30. July 2020.

Escoubet, C. P., Fehringer, M., and Goldstein, M. (2001). The Cluster Mission. *Annales Geophysicae*, 19(10/12):1197–1200.

Ferstl, R. (2016). Optimised Centroiding of Stars for Space Applications. *University of Vienna - Master's Thesis*, 1(11-23).

Forslund, A., Belyayev, S., and Ivchenko, N. (2007). Miniaturized digital fluxgate magnetometer for small spacecraft applications. *Journal of Astronautics*, 19(015202).

Hasiba, J. (2020). SMILE DPU Simulator User Manual. *Institut für Weltraumforschung - Oesterreichische Akademie der Wissenschaften*, 0.1(1).

Howell, S. B. (2006). *Handbook of CCD Astronomy*, volume 2. Cambridge University Press.

Hsu, T. and McPherron, R. L. (2003). Occurrence frequencies of IMF triggered and nontriggered substorms. *Journal of Geophysical Research: Space Physics*, 108(A7).

Hu, Y. Q., Guo, X. C., and Wang, C. (2007). On the ionospheric and reconnection potentials of the Earth: Results from global MHD simulations. *Journal of Geophysical Research Atmospheres*, 112(A7).

Huang, C., Le, G., and Reeves, G. D. (2004). Periodic magnetospheric substorms during fluctuating interplanetary magnetic field $\mathbb{B}_z$. *Geophysical Research Letters*, 31(14).

Janesick, J. (1987). *Charge-coupled-device charge-collection efficiency and the photon-transfer technique*, volume 26. Optical Engineering.

Jeszenszky, H. (2019). SMILE SXI EBOX DPU Design Reference. *Institut fuer Weltraumforschung, Oesterreichische Akademie der Wissenschaften*, (1).

Kozyra, J. and et al. (2014). Solar filament impact on 21 January 2005: Geospace consequences. *Journal of Geophysical Research: Space Physics*, 119(7).

Liu, Z. X., Escoubet, C. P., Pu, Z., Laakso, H., Shi, J. K., Shen, C., and Hapgood, M. (2005). The Double Star Mission. *Annales Geophysicae*, 23(8):2707–2712.

Loicq, J., Fleury-Frenette, K., Blain, P., and et. al (2018). Development of VUV multilayer coatings for SMILE-UVI instrument. *International Conference on Space Optics*, 1(0).

Lovett, R. A. (2011). What if the biggest solar storm on record happened today? https://www.nationalgeographic.com/news/2011/3/110302- solar-flares-sun-storms-earth-danger-carrington-event-science/. Accessed: 06 July, 2020.

Lucsanyi, D., Prod'homme, T., Smit, H., and et al. (2018). Pyxel – a novel and multipurpose Python-based framework for imaging detector simulation. *Conference: High Energy, Optical, and Infrared Detectors for Astronomy*, (1).

Marsaglia, G. and Tsang, W. (2000). The Ziggurat Method for Generating Random Variables. *Journal of Statistical Software*, 5,8(1-7).

Martin-Serrano, A. and Herrero, J. (2019). SMILE PLM Simulator: Software User Manual. *HVS EGSE Developement Team*, 3(2).

Mauk, B. H., Fox, N. J., Kanekal, S. G., Kessel, R. L., Sibeck, D. G., and Ukhorskiy, A. (2013). Science Objectives and Rationale for the Radiation Belt Storm Probes Mission. *Space Science Reviews*, 179(1).

Milan, S. E., Gosling, J. S., and Hubert, B. (2012). Relationship between interplanetary parameters and the magnetopause reconnection rate quantified from observations of the expanding polar cap. *Journal of Geophyiscal Research (AGU)*, 117(A3).

Miles, D. M., Clurzynski, M., Barona, D., and et al. (2019). Low-noise permalloy ring cores for fluxgate magnetometers. *Data Syst. (GI)*, 8(227-240).

Miyoshi, Y., Shinohara, I., and Takashima, T. (2018). Geospace exploration project ERG. *Earth, Planets and Space*, 70(1).

Moodya, I., Watkinsa, M., and Bell, R. (2017). CCD QE in the Soft X-ray Range. *e2v Technologies*.

Mukai, T., Nagai, T., Hoshino, M., Saito, Y., Shinohara, I., Yamamoto, T., and Kokubun, S. (2000). GEOTAIL observations of magnetic reconnection in the near-Earth magnetotail. *Advances in Space Research*, 25(7-8):1679–1683.

Pricea, G. J., Bruntona, A. N., and Beijersbergen, M. W. (2012). X-ray focusing with Wolter microchannel plate optics. *ScienceDirect*, 490(1-2).

Read, A. (2018). Orbits, Views and SXI Observations and Science Files.

Reimers, C., Ottensammer, R., and Kerschbaum, F. (2019). SMILE SXI IASW. *uVie IfA SW Architectural Design*, 1(0).

Rottinger, M. (1999). Charge-Coupled Devices. *TU-Wien - IUE*, 1(1).

Seelig, J. (2018). Characterization of the Institute's CCD - FLI PL 16803 (Bachelor Thesis). *University of Vienna - Institute for Astronomy*, 1(1).

Sembay, S. F., Sun, T., and Wei, F. (2015). PROPOSAL - SMILE: Solar Wind Magnetosphere Ionosphere Link Explorer. *ESA Executive Summary*, 1(1).

Sibeck, D. G., Allen, R., Aryan, H., and et al. (2017). Imaging Plasma Density Structures in the Soft X-rays Generated by Solar Wind Charge Exchange with Neutrals. *Space Sci Rev (Springer)*, 214(79).

SMILE, E. T. (2018a). SMILE Mission Status (ESA). In *Departement of Space and Climate Physics of University College London*, volume 1 of *SXI Team Meeting*, Dorking, United Kingdom. Mullard Space Science Laboratory.

SMILE, E. T. (2018b). Soft X-ray Imager (SXI) Programmatics Performance. In *Departement of Space and Climate Physics of Univseristy College London*, volume 1 of *SXI Team Meeting*, Dorking, United Kingdom. Mullard Space Science Laboratory.

Snowden, S. L., Egger, R., Freyberg, M. J., and et al. (1997). Rosat Survey Diffuse X-ray Background Maps II. *The Astrophysical Journal*, 485(125-135).

Soman, M. and Randalls, G. (2018). SMILE SXI CCD Testing and Calibration Event Detection Methodology [Open_SMILE_EDU_TN_01_2.0]. *The Open University - School of Physical Sciences*, 2(1).

Sul, B., Kong, L., and Zhang, A. (2018). Design and Simulation of Light Ion Analyzer for SMILE Satellite. *Journal of Astronautics*, 39((3):347-354).

Townsley, L. and Broos, P. (2008). The PSU CCD simulator: a Monte Carlo method for simulating X-ray CCDs. *Penn State University*, 1(1).

Turc, L., Fontaine, D., Escoubet, C. P., Kilpua, E. K. J., and Dimmock, A. P. (2017). Statistical study of the alteration of the magnetic structure of magnetic clouds in the Earth's magnetosheath. *Journal of Geophysical Research: Space Physics*, 122(3).

A. Appendix

The Appendix includes additional material and information. Firstly, a German summary (Zusammenfassung) of the thesis, which is required by the University of Vienna. Then, an overview of all figures, tables, and equations in this work. And lastly, a list of commonly used abbreviations.

A.1 Zusammenfassung

Der ESA Solar-wind-Magnetosphere-Ionosphere-Link-Explorer Satellite ist eine dreijährige Mission, welche die Wechselwirkung zwischen dem Magnetfeld der Erde und der solaren Strahlung beobachtet und deren globale Auswirkungen analysieren soll. SMILE wird vier Instrumente beinhalten: der soft X-ray imager (SXI), welcher die Magnetopause beobachten wird, eine UV Kamera (UVI), welche das Spitzenmagnetfeld beobachten wird, einen Magnetometer (MAG), der sowohl die Stärke als auch Richtung des lokalen Magnetfelds miessen wird, und den Light Ion Analyser (LIA), der die Geschwindigkeitsverteilung der Ionen von dem solaren Wind aufzeichnen wird. Diese Arbeit befasst sich mit den Datensimulationen, der on-board Analyse und der Event-Detektion durch den SXI. Die dazugehörigen Algorithmen wurden in Python und C geschrieben.

Der Datensimulator muss in Echtzeit laufen können, und gleichzeitig anpassungsfähig genug sein, um bestimmte Sonderfälle austesten zu können. Außerdem muss er modular aufgebaut sein, damit er ohne großen Aufwand bei zukünftigen Missionen adaptiert werden kann. Dafür müssen die Events, das Rauschen, und die Detektoreigenschaften leicht änderbar sein, aber dennoch eine Ausführungszeit aufweisen, die geringer als die erwartete Belichtungszeit ist (für SMILE: 4 Sekunden). Wegen der Natur von weicher Röntgenstrahlung, ist die Eventerkennung ein essenzieller Teil in der Datenverarbeitungskette. Diese Eventerkennung beginnt in der Front-End-Electronic (FEE). Hier wird wegen dem begrenzten on-board Speicherplatz lediglich ein 5×5 Pixel-Paket an die Data-Processing Unit (DPU) weitergegeben. Sowohl FEE als auch DPU folgen gewissen Regeln, mit denen unterschieden wird, ob es sich bei dem Signal tatsächlich um ein magnetosphärisches Event, Hintergrundröntgenstrahlung oder kosmische Strahlung handelt. Sobald das Signal als ein Event bestätigt worden ist, wird es zu dem On-Board-Computer (OBC) weitergeleitet und bis zum nächstmöglichen Download archiviert. Diese Arbeit soll als ein detaillierter Einblick in die SMILE-Mission dienen und die Struktur des Simulators DaSi, die daraus entstandenen Testdaten und die restliche vom Institut für Astronphysik der Universität Wien entwickelte Datenverarbeitungskette von SMILE aufzeigen.

A.2 List of Figures

1.1 SMILE Logo 3

2.1 Operation Modes 8
2.2 SMILE Flight Orbit 9
2.3 SMILE Spacecraft 10
2.4 SMILE Platform 10
2.5 SMILE Payload Module 11
2.6 Earth's Magnetosphere 12
2.7 Dungey Cycle - Substorm 16
2.8 Charge Exchange Process 18
2.9 Micro Pore Optics 19
2.10 Cosmic X-ray Background 21
2.11 Schematic: SXI CCD 23
2.12 Schematic: Soft X-ray Imager 23
2.13 Schematic: Ultraviolett Imager 25
2.14 Schematic: Magnetometer 27
2.15 Schematic: Light Ion Analyser 28

3.1 Simulated Dayside Magnetosphere 32
3.2 Schematic of CCD Pixel 33
3.3 CCD Gate Three-Phase Transfer 34
3.4 CCD Readout 35
3.5 Spectral Response 36
3.6 Work Flow of DaSi 39
3.7 Cosmic Ray Examples 47
3.8 Event Examples 53
3.9 Event Shapes 54
3.10 SXI Flat Field 56
3.11 DaSi Sample Frame 59
3.12 Random vs Numpy Generation 61
3.13 Random Number Generation Speeds 61

3.14 Random Number Histograms 63
3.15 Performance of Number Generators 64
3.16 Run time of Noise Methods 66
3.17 DaSi Local Noise 66
3.18 Binning 68
3.19 OU SMILE Science Frame 70
3.20 Histogram of OU and DaSi Image 71
3.21 StarSim Example 72
3.22 Pyxel Example 72

4.1 DaSi Cases: Images 77
4.2 DaSi Cases: Signal Comparison 78
4.3 Sun-facing Histograms and Profiles 78
4.4 AIRS Science Frame 79
4.5 DaSi Output ARIEL AIRS 79
4.6 NIRspec Image 80
4.7 DaSi Output ARIEL NIRspec 80

5.1 SXI Simulator Chain 82
5.2 DPU Memory Structure 84
5.3 IASW Structure 85
5.4 DPU State Machine. 85
5.5 FEE State Machine. 86
5.6 Data Processing: FEE and DPU 88
5.7 CCS Pool Viewer 92
5.8 CTI Background of Event 93
5.9 SXI Shapes 94
5.10 Data Chain Simulator Layout 96
5.11 Data Chain Simulator GUI 97
5.12 Raspberry-Pi Module 98
5.13 PLM SpaceWire Gateway Default Configuration 99
5.14 Run Time - Intel i7 Configuration 100
5.15 Run Time - Cortex-A72 Configuration 100

A.3 List of Tables

2.1 Science Modes 7
2.2 Orbit Parameters 8
2.3 Heliophysics Terms 13
2.4 SXI Instrument Parameters 24
2.5 UVI Performance Requirements 25
2.6 MAG Parameters 26
2.7 LIA Parameters 28

3.1 DaSi Configuration Settings 41
3.2 DaSi Configuration Settings (cont.) 42
3.3 SWCX Rates 52
3.4 DaSi .FITS Header 69

5.1 Event Data Packet 87
5.2 State Machine Transitions 89
5.3 CCD Operation Modes 91
5.4 Data Processing Simulator Commands 95
5.5 RAPIDsim Run Times 101

A.4 List of Equations

2.1 Charge Exchange Reaction 18
2.2 Charge Exchange: Earth Example 18
3.1 X-ray Intensity 30
3.2 Tangential Direction of Soft X-ray Peak 31
3.3 PPMLR-MHD Boundaries 31
3.4 PPMLR-MHD Boundaries - y component 31
3.5 PPMLR-MHD Boundaries - z component 31
3.6 X-ray Emissivity 31
3.7 CCD Noise 37
3.8 Meyer-Neldel Rule 37
3.9 CCD Gain 38
3.10 Cosmic Rate 53
3.11 Background X-ray Rate 53

A.5 List of Abbreviations

ARIEL Atmospheric Remote-sensing Infrared Exoplanet Large-survey

CAS Chinese Academy of Sciences

CCD charge-coupled device

CME coronal mass ejection

CTI Charge Transfer Inefficiency

DBS DPU Boot Software

DPU Data Processing Unit

EDU Event Detection Unit

ENAs energetic neutral atoms

ESA European Space Agency

EUV extreme ultraviolet

FEE Front End Electronics

FoV Field of View

FPGA Field Programmable Gate Arrays

GUI Graphical User Interface

IASW Instrument Application Software

ICU Instrument Control Unit

IMF Interplanetary Magnetic Field

IWF Institut fuer Weltraumforschung

LIA Light Ion Analyser

MAG Magnetometer

MCP micro-channel plate

MPOs micro pore optics

PF Platform

PLM Platform Module

PM Propulsion Module

PSF Point Spread Function

PUS Packet Utilization Standard

RAPIDsim RAspberry PI Data simulator

RMAP Remote Memory Acess Protocol

RSD Radiation Shutter Door

SCCT Standard Class Clone Trooper

SMILE Solar-wind-Magnetosphere-Ionosphere-Link-Explorer

SpW SpaceWire

SVM Service Module

SWCX solar wind charge exchange

SXI Soft X-ray Imager

TBC To Be Confirmed

TBD To Be Determined

TC Telemetry Commands

TM Telemetry Packets

UVI Ultraviolet Imager