Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Kristine Døsvik

Design Space Exploration for Hyperspectral Image Processing Pipeline in HYPSO Mission

Master's thesis in Electronics Systems Design and Innovation
Supervisor: Milica Orlandic
Co-supervisor: Mariusz Eivind Grøtte
July 2022

NTNU
Norwegian University of
Science and Technology

Kristine Døsvik

Design Space Exploration for Hyperspectral Image Processing Pipeline in HYPSO Mission

Master's thesis in Electronics Systems Design and Innovation
Supervisor: Milica Orlandic
Co-supervisor: Mariusz Eivind Grøtte
July 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Design space exploration for hyperspectral image processing pipeline in HYPSO mission

Kristine Døsvik

5.07.2022

Contents

Figures

Tables

Acronyms

ACE Adoptive Coherence/Cosine Estimator. 64, 71, 73, 77

ACE-R Adoptive Coherence/Cosine R Estimator. 16, 17

ASMF Adjusted Spectral Matched Filter. 16, 17, 64, 71

AUC Area Under the Receiver Operator Characteristics (ROC) Curve. vii, 11, 30, 36, 37

CEM Constrained Energy Minimization. 16, 17, 64, 71, 73, 77

CRD Collabrative Representation for hyperspectral anomaly Detection. 18, 19, 64, 75

ECEF Earth Centered Earth Fixed. 49

F-MGD Fast Morphological and Guided filters. 18, 20, 66, 71

FrFT-RX Fractional Fourier Entropy Reed-Xiaoli. 18, 19, 75, 77

GRX Global Reed-Xiaoli Detector. 18, 64, 75, 77

HYPSO Hyperspectral SmallSat for Ocean Observation. xi, xiii, 1, 3, 4, 6, 10, 31, 63–66, 79

ICA Independent Component Analysis. v, 14, 15, 29, 34, 64, 69

LRX Local Reed-Xiaoli Detector. v, 18, 19, 64, 75, 77

MAU Multi-Attribute Utility. 9, 23, 68–70, 72, 74, 76

MAUT Multi-Attribute Utility Theory. 8, 9, 23

MCC Matthews Correlation Coefficient. 11

MNF Maximum Noise Fraction. 14, 29, 34, 64, 69

NIR Near-Infrared. 5

NTNU Norwegian University of Science and Technology. xi, xiii, 1

PCA Principal Component Analysis. 14, 29, 33, 34, 64, 67, 69, 71, 73, 75, 77

RGB Red Green Blue. 3, 5

SAM Apectral Angle Mapper. 16, 17, 64, 66, 71, 73, 77

SNR Signal to Noise Ratio. vi, 31, 32

SVM Support Vector Machine. v, 20, 21, 69, 77

SVU Single Value Utility. v, vi, 9, 23–25, 34

VIS Visual. 5

!!!NOTATER!!! Norske sammendraget – må fikses! Kilder i introen. (kan vente) P4: Run time estimations - Radiometric calibration – vent til den er ferdig! - correlation matrix - CR method and not jacobi method
P7: Sørge for at kildene ser bra ut.

Abstract

The Hyperspectral SmallSat for Ocean Observation (HYPSO) is a Norwegian University of Science and Technology (NTNU) research project that aims to detect poisonous algae blooms in the ocean. HYPSO uses hyperspectral imaging from space to obtain information about the algae blooms. The idea is that fish farms can actively use HYPSO's data, potentially saving their fish from dying due to the poisonous algae.

This master thesis contributes to the HYPSO project by conducting a rough design space exploration that addresses possible ways to do hyperspectral data processing onboard the satellite. It is crucial to do data processing onboard the satellite because hyperspectral images can become so big that it will take a lot of time to download them Earth in raw format. In addition, the satellite has a budget when it comes to power, and downloading data from the satellite to Earth is quite a power hungry process.

By doing a rough design space exploration of onboard data processing of hyperspectral images, we get insight into what combinations of algorithms could be beneficial to include in the onboard data processing. However, the rough design space exploration conducted in this thesis is, indeed, rough, and the produced result should therefore only be considered guidelines to help further planning of the project. It is important to note that the results are far from hard facts.

This thesis includes run time estimations, accuracy estimations, and outputted data size estimations of different types of algorithms. Onboard data processing pipelines can consist of several algorithms, so the estimated run time, accuracy, and data output size of the onboard data processing pipelines are produced by combining the estimates of the algorithms in the pipeline. This thesis also includes evaluative models to compare onboard data processing pipelines.

A common factor for what seems to be promising processing onboard data processing pipelines is that they include algorithms that aim to correct erroneous effects caused by imaging tools or the imaging process. The design space exploration also shows that it can be interesting to do further research on using dimensionality reduction algorithms for data compression purposes.

Sammendrag

HYPSO er et NTNU forskningsprosjekt som har som mål å oppdage giftige algeoppblomstringer i havet. HYPSO bruker hyperspektral avbildning fra verdensrommet for å få informasjon om algeoppblomstringen. Tanken er at oppdrettsanlegg aktivt kan bruke HYPSOs data, og potensielt redde fisken deres fra å dø på grunn av de giftige algene.

Denne masteroppgaven bidrar til HYPSO-prosjektet ved å utføre en grov design utforskning som tar for seg mulige måter å utføre hyperspektral databehandling ombord på satellitten. Det er avgjørende å gjøre databehandling ombord på satellitten fordi hyperspektrale bilder kan bli så store at det vil ta mye tid å laste dem ned Jorden i råformat. I tillegg har satellitten et budsjett når det kommer til strøm, og å laste ned data fra satellitten til jorden er en ganske strømkrevende prosess.

Ved å gjøre en grov design tforskning av databehandling ombord av hyperspektrale bilder, får vi innsikt i hvilke kombinasjoner av algoritmer som kan være fordelaktige å inkludere i databehandlingen ombord. Den viktig å understreke at den grove design-romutforskningen utført i denne oppgaven er grov, og det produserte resultatet bør derfor kun betraktes som retningslinjer for å hjelpe videre planlegging av prosjektet. Den produserte dataen er langt fra harde fakta.

Denne oppgaven inkluderer estimeringer av kjøretid, nøyaktighetsestimeringer og resulterende datastørrelsesberegninger for ulike typer algoritmer. Databehandlingsrørledninger ombord kan bestå av flere algoritmer, så den estimerte kjøretiden, nøyaktigheten og resulterende datautdatastørrelsen til prosesseringskjedene for databehandling ombord produseres ved å kombinere estimatene for algoritmene i kjeden. Denne oppgaven inkluderer også evalueringsmodeller for å sammenligne ombord databehandlingskjeder.

En felles faktor for det som ser ut til å være lovende data prosessering ombord, er at de inkluderer algoritmer som tar sikte på å korrigere feilaktige effekter som er forutsaket av bildeverktøy eller prosessen av å ta bildet. Utforskningen av designene viser også at det kan være interessant å forske videre på bruk av dimensjonalitetsreduksjonsalgoritmer for datakomprimeringsformål.

Acknowlegements

I would like to thank particularly three people for the help they have given me when writing this thesis. My supervisor Milica Orlandic and cosupervisor Mariusz Grøtte have given me good technical inputs and resources to check out. I have had several good discussions about my work with them. However, I appreciate the most how they have brought me back on track when needed.

I will also like to thank Dennis Langer for his involvement in my thesis. He has enlightened me about where I can find resources and what resources exist. Whenever I have met technical challenges, he has been open to discussing them with me and has generally been available for a casual conversation about my thesis. Consequently, the work has probably been more manageable than it could be, and definitively more fun, so I am very grateful.

Chapter 1

Introduction

1.1 The HYPSO Project and Problem Description

Norway's coastline stretches all the way from the south of the country to the north. Throughout Norway's history, easy access to the ocean caused fishing to play a major role in economy and tradition. Today, fish is still one of its most important resources. In addition to wild fish, we use fish farms to exploit the ocean resources without causing a drain on wild fish. Unfortunately, there exist certain kinds of algae that deplete oxygen solved in the water. These algae kill all the oxygen dependent species, including fish. For fisheries this can in the worst case mean loss of all resources (fish) and monetary income. The fishing industry in Norway have lost billions of kroner due to these deadly algae.

At NTNU there is a research project that addresses this issue. The project is called HYPSO and consist of students, researchers, professors and other employees of NTNU. The objective of the project is to monitor the deadly algae in the Norwegian coastal ocean, preventing the loss of farmed fish, and money. The idea is that by knowing where there are deadly algae, and not, the fish farmers can locate the fish farms accordingly.

The method HYPSO uses to discover and monitor the deadly algae is by taking hyperspectral images and check if the spectral signatures of deadly algae are present.

Currently, HYPSO has launched one satellite called HYPSO-1. HYPSO-1 is the first launched satellite in a series of satellites. HYPSO-1 can be considered a proof-of-concept satellite, and the next generation of satellite will use experience from HYPSO-1 to improve the next versions.

The HYPSO-1 satellite do data processing of hyperspectral images onboard the satellite. This is necessary as hyperspectral images can become substantially large. A substantially large image will require a lot of memory storage, a lot of time to download the image to earth, and a lot of power to download the image to Earth. Data processing reduce the data size, which is very important as it reduces the time and power used to download the image to the Earth. It is therefore important to carefully consider how to do onboard data processing.

This master thesis conducts a design space exploration of possible onboard data processing pipelines. The result of this thesis will give insight into what processing is likely to be promising for on board data processing. However, the design space exploration uses rough estimates so it is important to be skeptical when evaluating the results.

The result of the design space exploration can be used by several space missions. However, in this thesis we favor HYPSO, meaning that we use parameter values that are typically used by HYPSO.

1.2 Outline

First, we will motivate the task in chapter 2. Than we explain the theory used in this design space exploration in chapter 3, before elaborating on the implementation details in chapter 4. Chapter 6 displays and discuss interesting outputs of the design space exploration, and chapter 7 provides a conclusion of this master thesis.

Chapter 2

Motivation

2.1 Technical Motivation

As described in the introduction, the HYPSO mission takes hyperspectral images , which can have a large data size. While an everyday Red Green Blue (RGB) imager uses only three spectral values to represent an image, a hyperspectral imager can collect spectral samples up to over a thousand different electromagnetic wavelengths [1]. So compared to everyday RGB images, a hyperspectral image can become massive.

The limited time to download data in an orbital pass is a problem when handling large hyperspectral images. The satellite can only communicate with the Earth when passing the coverage area of the ground stations, which for the HYPSO-1 satellite is estimated to last typically 7,5 minutes in the internal document "Mission design, operations scheduler and payload trade-offs for HSI v6, hypso-1" [2].

Power consumption also restricts the amount of data transferred from the satellite to the Earth. The HYPSO satellite has only a certain amount of energy for disposition in every orbital pass, and downloading data from the satellite drains a lot of energy compared to other power demanding activities [3]. To ensure the satellite does not run out of energy, the power budget of HYPSO-1 allows only 10 minutes of Earth communication for each orbital pass [3].

Onboard data processing can reduce the amount of data that is crucial to download to Earth and consequently lessen the needed communication time with Earth and the power usage for downloading data.

However, there are many possibilities for doing onboard data processing. In addition to using different processing algorithms, we can combine them in different ways, use different algorithms, and choose between software and hardware implementations. The numerous possibilities motivate the design space exploration conducted in this thesis.

There are a lot of possible ways of doing onboard data processing. Nevertheless, not all type of processing is assumed to be equally good. There are mainly two factors that determine if the data processing is sound:

1. how good accuracy does the outputted data have,
2. how long time it takes to complete the data processing, and
3. how much data is necessary to download to the Earth.

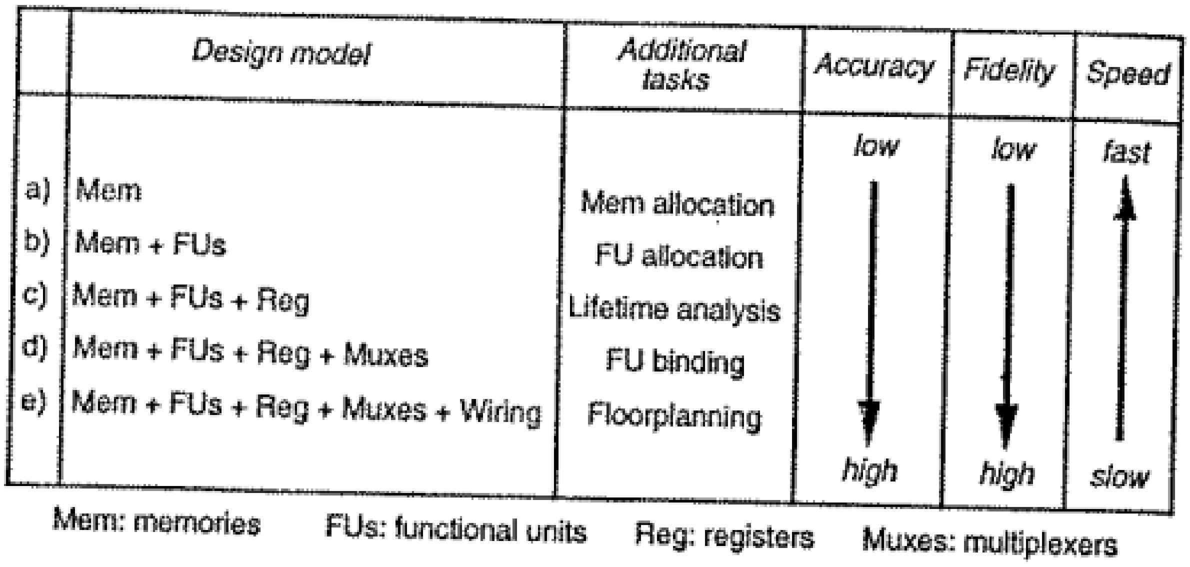
It would be pointless to apply processing that makes the quality of the hyperspectral data so bad that the data is no longer trustworthy. It would also be suboptimal to execute data processing that takes a long time to complete or does not reduce the data size.

2.2 Planning Motivation

The flow of a project is typically divided into different phases. The early phases concern planning, and the latter stages implementation, maintenance, and disposal. The planning stages often include a design space exploration where a lot of different mission concepts are examined [4]. This master thesis concerns the conduction of a design space exploration, helping further planning of the HYPSO mission.

The goal of doing the design space explorations is to provide a tool for the project managers to make a scientifically justified decision about what path the project should follow. If the project managers do not make good and thoughtful decisions in the start phase of the project, there is more chance of making big mistakes, which can be costly and time-demanding. Good planning, on the other hand, increases the probability that the project development runs smoothly. [5].

When making decisions based on estimates, it is important to remember that the estimates can be wrong, and we can draw a wrong picture. There is a trade off between time usage and estimation accuracy. The more relationships incorporated into the estimation, the more accurate the estimations are, but the longer it will take to make the estimates [6]. Figure 2.1 shows an example of the time and estimates accuracy for a hardware estimation problem. The figure illustrates how the time to estimate the algorithms increases when more factors are taken into account (accuracy increases). The same goes for other kinds of estimations.



	Design model	Additional tasks	Accuracy	Fidelity	Speed
			low	low	fast
a)	Mem	Mem allocation	↓	↓	↑
b)	Mem + FUs	FU allocation			
c)	Mem + FUs + Reg	Lifetime analysis			
d)	Mem + FUs + Reg + Muxes	FU binding			
e)	Mem + FUs + Reg + Muxes + Wiring	Floorplanning			
			high	high	slow

Mem: memories FUs: functional units Reg: registers Muxes: multiplexers

Figure 2.1: "Typical estimation models and associated accuracy, fidelity and speed of estimation [6].", for a hardware design problem.

The perfect estimations are the ones that are good enough to make the right decisions and fast enough to evaluate a lot of different solutions [7].

Chapter 3

Background and Method

3.1 Hyperspectral Imaging

In this thesis, the expression 'onboard data processing' refers to processing a hyperspectral image onboard a satellite. This section briefly explains what a hyperspectral image is and strategies to capture hyperspectral images.

3.1.1 Hyperspectral Images

The similarity between hyperspectral and RGB images is that they are snapshots of a spatial area. On the other hand, the difference is the number of wavelengths that make up the image. An RGB image is made up of the three electromagnetic wavelengths corresponding to the colors red, green, and blue. Hyperspectral images are made up of several wavelengths, usually in the Visual (VIS) and Near-Infrared (NIR) spectrum, about 400 to 1000 nm [8–10].

If hyperspectral images capture a broad range of spectral wavelengths with a sufficient resolution, they can be used to detect, classify and identify objects. This is possible as each object has unique reflection and absorption properties when illuminating the object with wavelengths of the electromagnetic spectrum. An object's reflectance and absorption properties are directly related to the object's physical and chemical composition [1].

Figure 3.1 illustrates that different materials have a distinct spectral signature, or reflectance, where the blue graph shows the spectral signatures of deep water, the green shows the spectral signature of a spruced needle, the yellow the spectral signature of sand, and the red the spectral signature of dry grass [11].

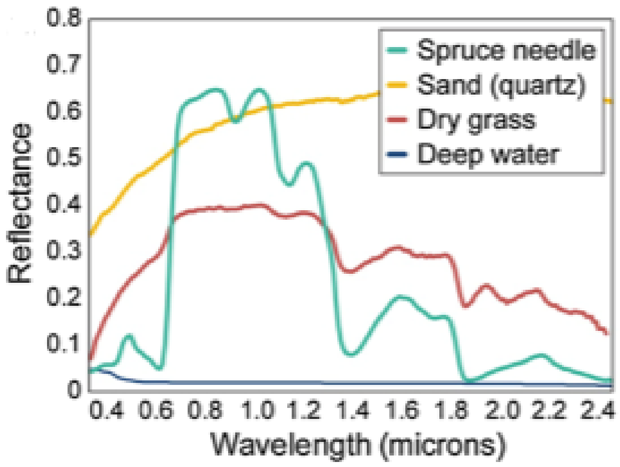


Figure 3.1: Spectral signatures of a spruce needle, sand, dry grass, and deep water. X-axis denotes the wavelength in micrometers, and y-axis shows the amount of reflection of each wavelength [11].

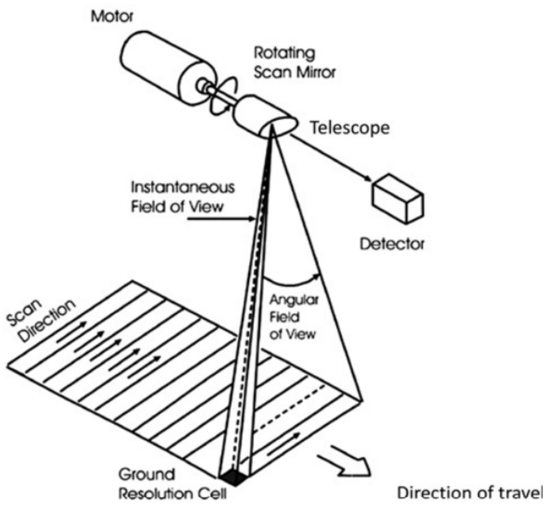
Hyperspectral images can be seen as three dimensional cubes. There are two spatial dimensions representing the snapshot area, and a third spectral dimension representing the number of wavelengths used to represent each spatial pixel.

3.1.2 Capturing Hyperspectral Images

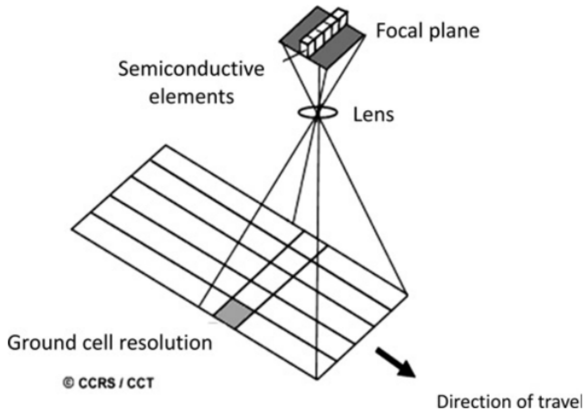
The tools that capture hyperspectral images are hyperspectral imagers. According to Keith, D., there are two basic kinds of hyperspectral imagers, whiskboom and pushbroom scanners [12].

The whiskbroom scanner takes pictures of the earth by sweeping the surface perpendicular to the flight direction. The pushbroom scanner captures hyperspectral images by taking a series of two dimensional snapshots in the along track direction. The two dimensional snapshots consist of one spatial dimension and one spectral dimension. Figure 3.2a shows a whiskbroom scanner, and figure 3.2b shows a pushbroom scanner.

The HYPSO mission uses a pushbroom scanner. We address the hyperspectral snapshots as frames, in figure 3.3 the number of frames in the hyperspectral cube is denoted as M. Every frame/hyperspectral snapshot consists of a predefined number of spatial pixels, denoted as N in figure 3.3. The spatial pixels in a frame are made up of measurements of spectral values, whose number is represented by K in figure 3.3.



(a) A whiskbroom scanner [12].



(b) A pushbroom scanner [12].

Figure 3.2: Hyperspectral image scanners.

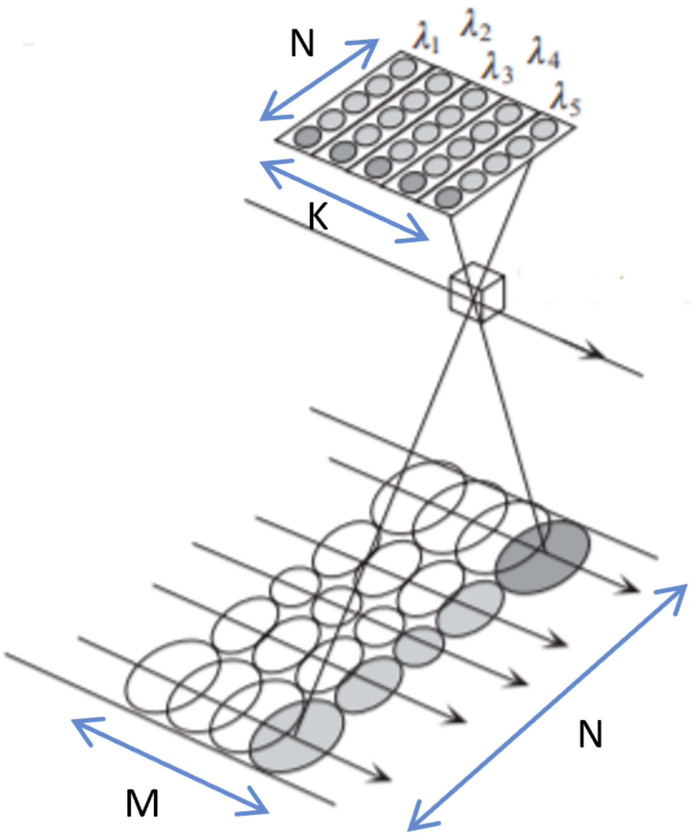


Figure 3.3: How a push broom scanner collects a hyperspectral image, and stores a spatial line as a two-dimensional frame [13].

3.2 Design Space Exploration

According to João M.P.Cardoso, “Design Space Exploration (DSE) is the process of finding a design solution, or solutions, that best meet the desired design requirements, from a space of tentative design” [14].

Design space exploration is getting an overview of different solutions to a problem, while a decision problem is choosing what solution to use.

It is normal to explore the alternatives to a decision problem by conducting a design space exploration. The design space exploration makes it possible to evaluate and compare different designs in terms of predefined requirements. By basing a decision on a design space exploration, it is likely that the decision appears more thoughtful [5].

When facing a decision problem, we use models to evaluate the designs. The models can either be mental models or constructive models. Mental models are evaluating models generated in the decision makers’ heads. Mental models can be biased according to each decision maker’s values, experience, and prejudice, which might be hard to understand from another person’s point of view. A constructive model is a mathematical model that deterministically produces a measure of goodness for each design. We will obtain a more objective and transparent evaluation of the different solutions using constructive models. Nevertheless, if the constructive model diverges too much from any of the decision makers’ mental models, the constructive model loses trust and can be discarded [5].

Figure 3.4 illustrates the process of making a decision. The first step is getting an overview of possible designs, in other words, conducting a design space exploration. The next step is using models to evaluate the various designs. Based on the evaluation of the designs, we might reconfigure our design space or models and repeat the step of design evaluation. After an undetermined number of model and design reevaluations, we arrive at a decision [5].

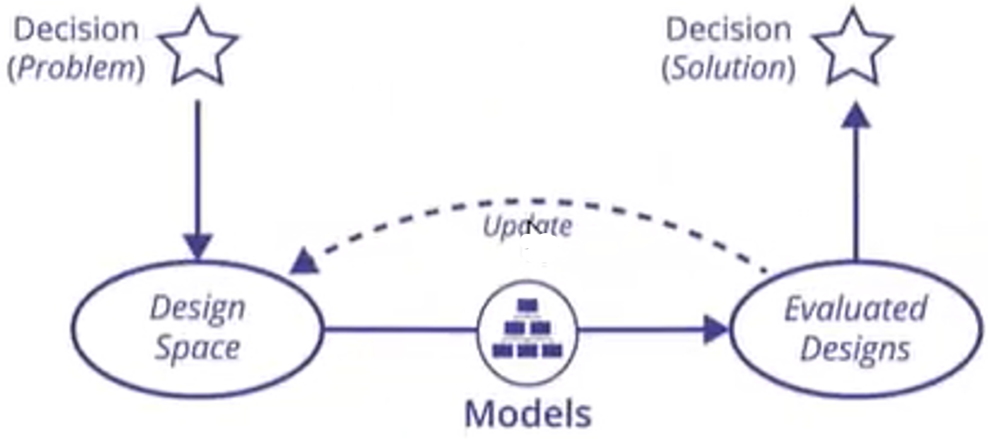


Figure 3.4: The process going from a decision problem to a decision solution [5].

3.2.1 Evaluation Model

There are plenty of models which will evaluate the designs differently. What model is suitable for which decision problem depends on the nature of the decision problem. Ross et al. mention several possible evaluation models, including the Multi-Attribute Utility Theory (MAUT) used in this design space exploration, in the article “Aligning Perspectives and Methods for Value-Driven Design” [15].

The MAUT methodology perceives value in a manner that is compatible with the onboard data processing problem. In the MAUT methodology value is interpreted as “the aggregation of (non-monetary) benefits relative to the monetary cost of obtaining those benefits" [15]. In the onboard decision problem value can also be perceived as non-monetary benefits, namely small outputted data size, and high data quality. Consequently, the MAUT methodology is fitting for a decision problem like the onboard data processing problem.

3.2.1.1 Multi-Attribute Utility Theory

The MAUT methodology decomposes the designs into attributes and evaluates each attribute separately. The final Multi-Attribute Utility (MAU) score is then produced by aggregating the attribute scores with a combination rule. The designs are compared through the MAU scores [15] [16].

Equation (3.1) shows a possible combination rule for producing an MAU score of n attributes,

$$MAU = \sum_{i=1}^{n} f_i(x_i), \tag{3.1}$$

Where x_i is the state of the ith attribute, f_i is the SVU function to the ith attribute multiplied by the attributes weighting factor. The combination rule shown in (3.1) does not take uncertainty into account, but there are combination rules that do [16].

The SVU functions of the attributes assign the attribute an SVU score between 0 and 1, depending on the attribute state. An SVU score of zero indicates the least acceptable value, while a score of one indicates the best case scenario or even better. If the input value score is below the least acceptable value, the sample is declared infeasible and discarded [15]. Figure 3.5 illustrates an example of a SVU function.

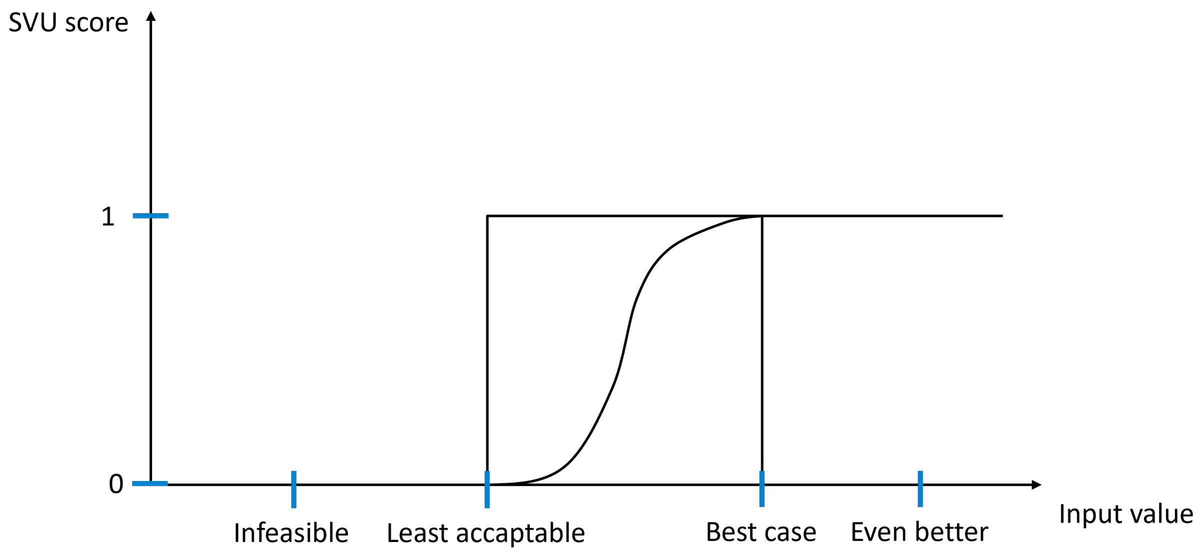


Figure 3.5: An example of an SVU graph.

The cost metric does not need to be included as an attribute. If the cost is excluded, it is possible to do a benefit-cost comparison [15].

3.3 Estimations

A design space exploration is typically conducted in the start phase of a project to decide the implementation details. There are probably numerous implementation alternatives where few or none are implemented at this stage of the project. Even though the alternatives are not implemented, it is possible to compare them by creating estimations that reflect the designs' features. This section explains the estimation of the onboard data processing design's processing time, accuracy, and outputted data size.

3.3.1 Estimating Processing Time

The estimated processing time of one onboard data processing pipeline is the sum of the estimated processing times for each pipeline stage.

It is normal to use the big oh notation to estimate the run times of algorithms [17]. The big oh notation displays the algorithms' complexity and hides constant run time factors.

Another type of theoretical run time estimation is analyzing the algorithms' instructions [17]. In this kind of run time estimation, we estimate the number of instructions and each instruction's time usage. In the book "Foundations of Computer Science: C Edition," Ullman argues that using the big oh notation is a better run time estimate than analyzing the number of algorithm instructions because [17]:

1. run time depends on the computer that runs the algorithms. The number of executed instructions per second is different for different computer platforms. Consequently, algorithms can run up to 1000 times faster by just switching computer platforms. And,
2. compiling algorithms with different compilers affect the resulting run time of an algorithm, as the compilers can generate different amounts of machine instructions.

Following the line of reasoning, analyzing the number of instructions in an algorithm "are at best approximations to the truth," as the constant run time factors in instruction analytic estimations can be modified broadly by compilers and computer platforms [17]. Nevertheless, if the run time estimation is based on the number of operations rather than the complexity, one can distinguish implementations with the same complexity to a higher degree. Operation counts give insights into if an algorithm is, for example, likely to be twice as fast or ten times as fast as another. However, it is important to notice that the operation count only denotes the probability of the implementation being faster or slower. As elaborated previously, platforms and compilers can affect the implementations in ways that are difficult to foresee.

In an operation count, we make assumptions about how long it takes to perform different operations and how the assumed operation platform handles memory and overhead operations. By clearly stating all the assumptions, it is more likely that the creators and others can interpret the data meaningfully.

HYPSO-1 has already conducted a small design space exploration for onboard data processing.

In the HYPSO-1 design space exploration, algorithmic processing times are estimated as a linearization of the inputted data size and a run time sample from a performance study [3] [2].

The benefit of the method used for HYPSO-1 is that it is fast, and the drawback is that it assumes the run times scale linearly with the hyperspectral cube size, which is not always the case.

3.3.2 Estimating Outputted Data Size

The hyperspectral data is passed from processing algorithm to processing algorithm, which will reduce, increase or not affect the data size. Similar to run time estimation, data output is estimated is a step-wise fashion. The final data size output of the pipeline can be affected by how the different processing algorithms (pipeline stages) have previously changed the data size. It can therefore be necessary to analyze how the data is changed in all individual processing stages to estimate the final outputted data size correctly.

Many algorithms have deterministic outputted data sizes. Consequently, it is possible to state an algorithm's outputted data size by understanding its objective. Nevertheless, in some cases, it is not possible to determine the outputted data size by understanding the purpose of the algorithm. Then, the reduction or increase of the outputted data size, compared to inputted data size, can be estimated according to the decrease or increase factor in typical performance studies.

3.3.3 Estimating Accuracy

It is possible to obtain accurate estimates of a processing pipeline by looking into algorithms' objectives and performance studies.

It is important to consider the objective of the algorithms as they potentially affect the output accuracy of some algorithms. An example of an algorithm that can greatly impact target detection algorithms but not anomaly detection algorithms is smile and keystone correction. Smile and keystone correction reverse wavelength errors caused by the imaging tool. Such a wavelength correction can be critical for target detection algorithms and insignificant for anomaly detections. Anomaly detection would probably detect anomalies even though the measured spectral signatures are slightly off. On the other hand, target detection is more likely not to see targets if the spectral signatures are too twisted.

The design space exploration can form onboard processing pipelines that combine algorithms in a manner not usually seen. It can therefore be hard or impossible to find performance studies for the different onboard data processing pipelines. It is far more likely to find performance studies of the individual algorithms in the processing pipelines. A simple way to estimate the accuracy can be to estimate the accuracy of each algorithm independently and then combine them by multiplication. In general, the described method does not take into account that some algorithms affect other algorithms' output to different degrees. But, it is possible to include such dependencies by making reasonable assumptions about what effect the various algorithms have on each other.

A challenge of estimating the accuracy through already conducted performance studies is that the accuracy performance of algorithms can be expressed through different data quality metrics. In the article "A reconfigurable multi-mode implementation of hyperspectral target detection algorithms", Bos̆ković et al. use visibility and Matthews Correlation Coefficient (MCC) to quantify the quality of target detection algorithms [18]. Gundersen, on the other hand, uses AUC to evaluate the quality of anomaly detection algorithms in his master thesis "Hardware-Software partitioned implementation of an autoencoder-based hyperspectral anomaly detector" [19].

Even though evaluating specific algorithms with specific evaluation metrics might be preferable, we should strive to find a common quality metric. Different quality metrics measure different kinds of quality and can therefore not easily be compared in a meaningful manner.

3.3.3.1 AUC

The AUC score is a quality score for classifying algorithms and tells how good the algorithm is at distinguishing classes. AUC is defined in the range between 0 and 1. An AUC score of 1 means that the algorithm can classify objects perfectly, a score of 0.5 means that it classify 50% of the objects correctly, and 0 means that it classifies all the object wrong [20].

3.4 Processing Modules

There are numerous ways of processing hyperspectral images onboard a satellite. The processing can consist of several processing modules applied subsequently, forming a processing pipeline. This section briefly explains the different processing modules assessed in this design space exploration.

3.4.1 Binning

The purpose of a binning algorithm is to transform variables into a smaller set of groups and is, therefore, a kind of lossy compression. If the number of samples is divisible by the binning factor, all the bins are equally sized. If not, the last bin contains the leftover samples [21]. Figure 3.6 illustrates the process of binning.

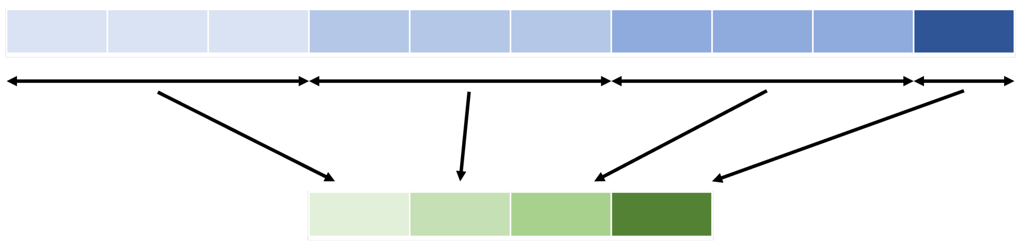


Figure 3.6: Binning ten values with a binning factor of three.

In spatial binning, we bin spatial values, while in spectral binning we bin spectral values.

3.4.2 Bad Pixel Detection and Correction

A bad pixel is a pixel that behaves as an anomalous in an image because of faults in the camera [22]. Figure 3.7 shows how bad pixels can appear in images.

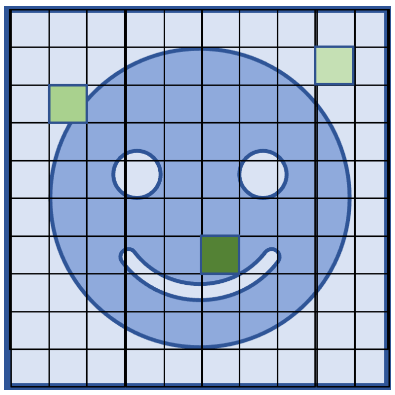


Figure 3.7: Three bad pixels in an image.

Bad pixel mitigation algorithm aims to detect and correct bad pixels. The correction consists of estimating a suitable pixel value.

The following subsections explains the concepts of the bad pixel detection and correction algorithms that are included in this design space exploration.

3.4.2.1 Bad pixel detection: Statistical Threshold Check

In the Statistical threshold method we calculate the mean and variance of a set of neighbor pixels. If the pixel in question has a value diverge more from the mean than two times the variance, we assume that the pixel is a bad pixel [22].

3.4.2.2 Bad pixel detection: Correlation Check

In this detection algorithm we assume that neighbour pixels correlate well. We therefore compute the correlation between a pixel and its N number of neighbours. If there are a lot of bad correlations we conclude that the pixel in question is a bad pixel [22].

3.4.2.3 Bad pixel detection: Mean Threshold Check

By assuming that pixels correlate well with their neighbors, as in the correlation check, we can form another bad pixel detection algorithm that is based on computing averages. It is reasonable to believe that if pixel correlates well with their neighbors, they will probably not differ much in value from their neighbors. With this in mind, we can form a bad pixel detection algorithm that checks whether a pixel is bad by comparing it to its neighbor's average value. If the neighbors' average value differs too much from the pixel in question, we assume the pixel is a bad pixel.

3.4.2.4 Nearest Neighbour Replacement

Nearest neighbour replacement is the simplest correction method and consist of copying the value of one of its neighbours [22].

3.4.2.5 Mean Replacement

Mean replacement method replace the bad pixel's with the mean value of its neighbors [22].

3.4.2.5.1 Median Replacement

The median replacement method is similar to the mean replacement method, but uses the neighbours median [22].

3.4.3 Smile and Keystone Correction

Hyperspectral images suffer from wavelength shifts and band to band misregistration that arise from properties of hyperspectral sensors [23]. Figure 3.8 shows the difference between an ideal image and the obtained hyperspectral image.

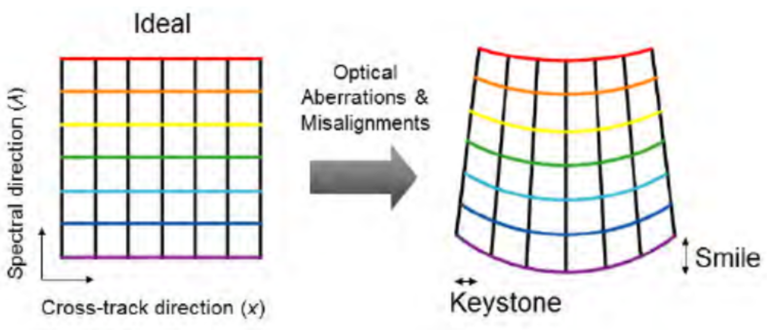

Ideal
Spectral direction (λ)
Cross-track direction (x)
Optical Aberrations & Misalignments
Smile
Keystone

Figure 3.8: An ideal image and an image that suffers from smile and keystone properties [23].

3.4.4 Radiometric Calibration

Hyperspectral imaging tools measure light by generating a current corresponding to the light intensity. If there is no light at a given wavelength, the imaging tool should generate zero current for the relevant light sensors. However, imaging tools suffer from something called dark current. When there is no observable light, the sensors still generate some current, which is called dark current. The dark current functions as random intensity offsets and are undesirable, and radiometric calibration aims to reverse the dark current offsets [24]. In addition, radiometric calibration maps the raw sensor output of digital number to physical units of radiance given in e.g. Watt (W) per square meter (m^2) per steradian (sr) per wavelength (nm) $\left[\frac{\mathrm{W}}{\mathrm{m^2\,sr\,nm}}\right]$.

3.4.5 Dimensionality Reduction

Dimensionality reduction aims to reduce the dimensions representing spatial pixels by exploiting correlation in the spectral domain. If there is a high correlation among the spectral bands, the spatial pixels can be represented more compactly without losing too much information [1].

According to Ibarrola-Ulzurrun et. al., there are three widely used dimensional reduction algorithms [25],

- Principal Component Analysis (PCA),
- Maximum Noise Fraction (MNF) Transform, and
- ICA.

In this design space exploration, we include these three algorithms.

3.4.5.1 PCA

In principal component analysis, we represent the data with a new set of vectors. The new vectors are made according to the samples' variance, where some label lots of variations while others almost none. The more variance a vector label, the more information it contains. Therefore, it is safe to discard the vectors that label the least amount of variance, thereby reducing the dimensions [1].

3.4.5.2 MNF

The Maximum Noise Fraction algorithm resembles PCA, but does dimensional reduction according to noise and not variance [26] [27].

3.4.5.3 ICA

Independent Component Analysis aims to decompose a measured signal into its independent sources [28]. Figure 3.9 illustrates the concept of ICA.

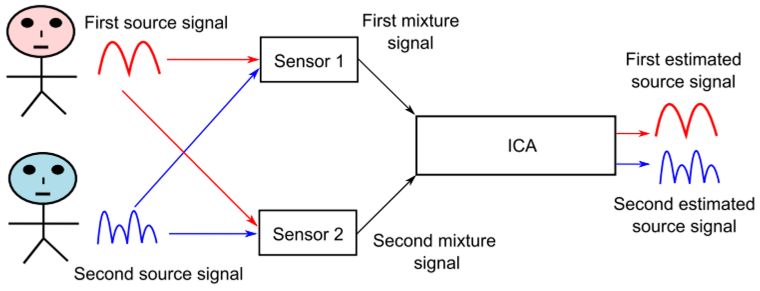
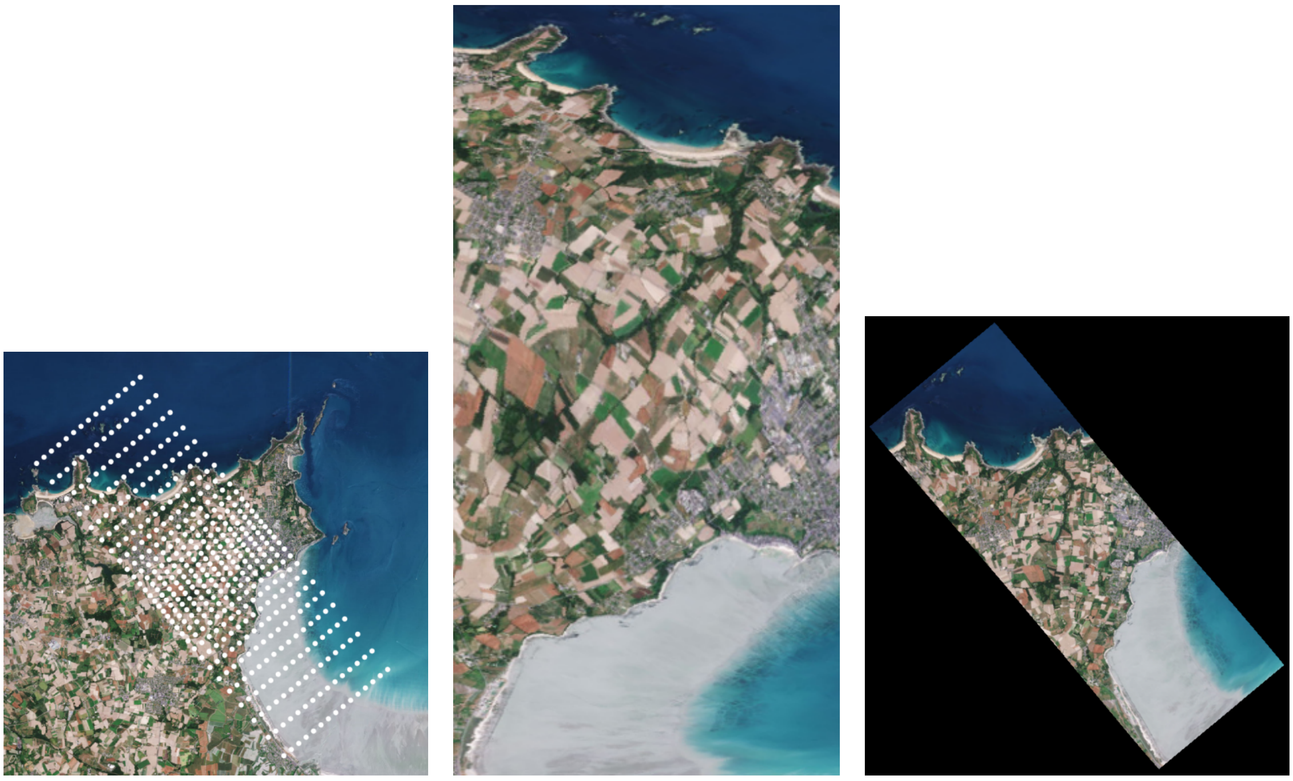


Figure 3.9: ICA demixes two measured sound signals and obtain estimates of the sound sources [28]

3.4.6 Georeferencing and Geometric Registration

Georeferencing connects the spatial pixels in a hyperspectral image to locations on Earth. Using the satellite's position and attitude records makes it possible to estimate the spatial pixels' longitude and latitude coordinates [29].

Geometric registration corrects the spatial distribution of the image according to the estimated longitude and latitude coordinates. The resulting output is a new hyperspectral image having the true proposition of the Earth [29]. The figures 3.10a, 3.10b and 3.10c illustrate how geometric correction corrects geometric distortions in images taken of the Earth from a high altitude, while the imager is doing irregular sampling.



(a) How samples are collected from earth when the satellite is pitching, leading to an irregular sampling pattern [29].

(b) The raw image containing geometric distortions obtained by collecting the samples shown in figure 3.10a [29].

(c) Georeferenced and geometric corrected image of the raw image 3.10b.

Figure 3.10: Image series illustrating geometric registration

3.4.7 Target Detection

The purpose of target detection is to detect objects with predefined spectral signatures [18]. Figure 3.11 illustrates a scene observed by a satellite, and as seen in figure 3.12 the different elements in the scene have different spectral signatures. the satellite can therefore detect elements by comparing the observed spectral data to their known spectral signatures.

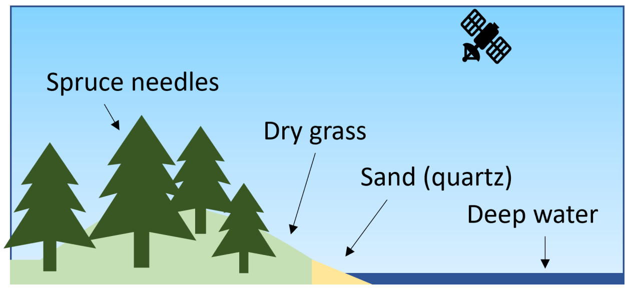


Figure 3.11: A satellite observing a landscape containing spruce needels, sand, dry grass and deep water.

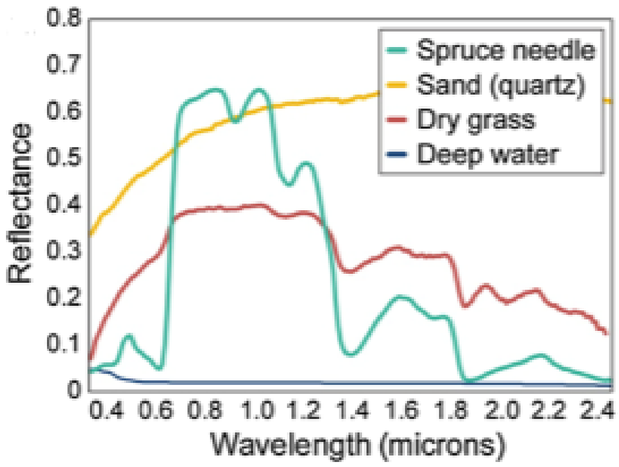


Figure 3.12: Spectral signatures to spruce needels, sand, dry grass and deep water [11].

The article "A reconfigurable multi-mode implementation of hyperspectral target detection algorithms" by Đorđije Bos̆ković et. al. mentions the target detection algorithms,

- Apectral Angle Mapper (SAM),
- Constrained Energy Minimization (CEM),
- Adoptive Coherence/Cosine R Estimator (ACE-R), and
- Adjusted Spectral Matched Filter (ASMF)

among others [18]. We include the listed algorithms in our design space exploration.

3.4.7.1 SAM

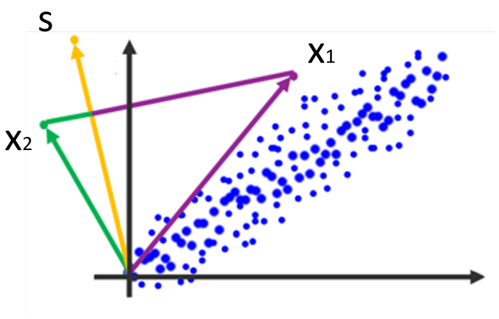
The SAM algorithm compute the vector angle between a measured signal **x** and the target signal **s**, by utilize the mathematical identity,

$$\frac{\mathbf{s}^{\mathrm{T}} \cdot \mathbf{x}}{|\mathbf{s}| \cdot |\mathbf{x}|} = \cos(\theta). \tag{3.2}$$

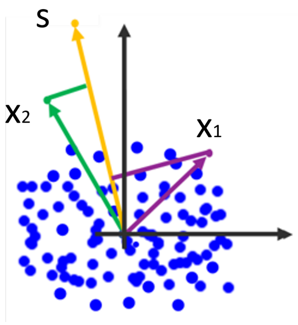
The smaller the angle between $\mathbf{x}$ and $\mathbf{s}$, the more confident the SAM target score [18].

3.4.7.2 CEM

The CEM algorithm uses projection of whitened samples to compute target detection scores [18]. By whitening samples we mean removing the sample correlation. Figure 3.13 illustrate how the target detection scores based on projection differ when whitening samples and not.



(a) Target detection when data samples are not whitened.



(b) Target detection when the data samples are whitened.

Figure 3.13: Target detection by projection with and without whitened samples.

Figure 3.13a shows that even though sample $\mathbf{x}_2$ is more likely to be a target than $\mathbf{x}_1$, they obtain the same target score.

On the other hand, figure 3.13b shows that by whitening the samples first, the projection of $\mathbf{x}_2$ and $\mathbf{x}_1$ onto the target $\mathbf{s}$ differs. Hence, CEM computes more realistic target detection scores.

3.4.7.3 ACE-R

The ACE-R algorithm is a modification of the SAM algorithm. ACE-R, as SAM, computes the angle between a sample and the target, but whitens the samples first using the correlation matrix [18].

3.4.7.4 ASMF

The ASMF is a modification of CEM. It computes a CEM target score but adds a non-target pixel suppression factor that aims to suppress or amplify different spectral features [18].

3.4.8 Anomaly Detection

Anomaly detection is a kind of unsupervised target detection. The difference between target detection and anomaly detection is that target detection looks for predefined features in an image, while anomaly detection looks for distinct features.

Abnormal features can be interesting to investigate, as we might have discovered something special. Figure 3.14 shows a scene observed by a satellite that mainly consist of dry grass, but also one spruce tree. The spruce tree can therefore be categorized as an anomaly.

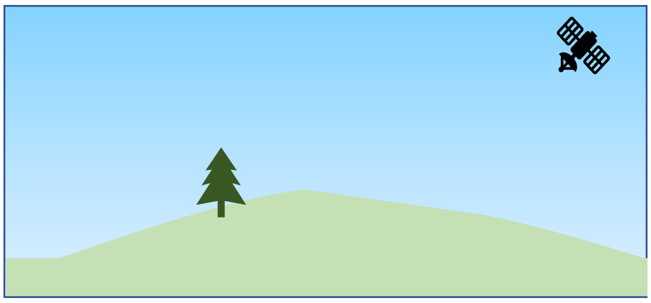


Figure 3.14: A satellite observing a landscape containing dry grass and spruce needles.

The master thesis "Hardware-Software partitioned implementation of an autoencoder-based hyperspectral anomaly detector" by Gundersen mentions, among others, the anomaly detection algorithms [30],

- Global Reed-Xiaoli Detector (GRX),
- LRX,
- Fractional Fourier Entropy Reed-Xiaoli (FrFT-RX),
- Collabrative Representation for hyperspectral anomaly Detection (CRD), and
- Fast Morphological and Guided filters (F-MGD).

In this design space exploration we assess a modified version of GRX and LRX, and the other listed anomaly detection algorithms. The modification of GRX and LRX is that we compute an anomaly score based on the correlation matrix and not the proposed covariance matrix.

3.4.8.1 GRX

The GRX algorithm uses the Mahalanobis distance with the correlation matrix to compute an anomaly score, as shown in (3.3).

$$GRX = (\mathbf{x} - \boldsymbol{\mu})^T \hat{R}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \tag{3.3}$$

3.4.8.2 LRX

The LRX anomaly detection algorithm is similar to the GRX, but uses only a sub area of the image to compute the anomaly detection score. The mean and correlation matrix needed in the computation are therefore local for the sub area. Figure 3.15 illustrate the sub area used to assign an anomaly score for the pixel marked as a black square.

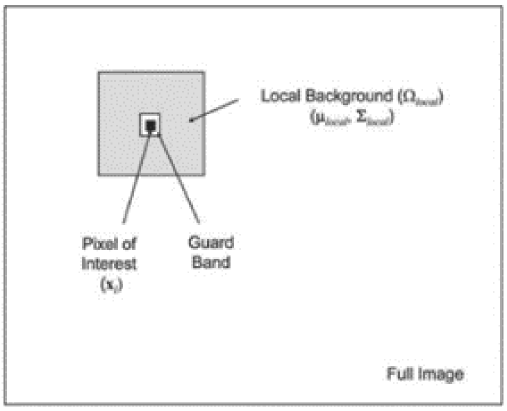

Local Background (Ω_{local})
(μ_{local}, Σ_{local})
Pixel of Interest (x_i)
Guard Band
Full Image

Figure 3.15: The local area where LRX is computed [1].

3.4.8.3 FrFT-RX

FrFT-RX is a RX anomaly detection combined with some preprocessing. The preprocessing consist of representing each pixel as a mixture of its spectral values and its Fourier transform (therefore the name fractional Fourier transform). Figure 3.16 shows two graphs, the right shows the spectrum of two background pixels (red and blue) and one anomaly pixel (yellow), while left graph show the mixed representations of the same two background pixels and the anomaly. As shown in figure 3.16 changing the domain of the pixels can increase the difference between background pixels and anomalies [31].

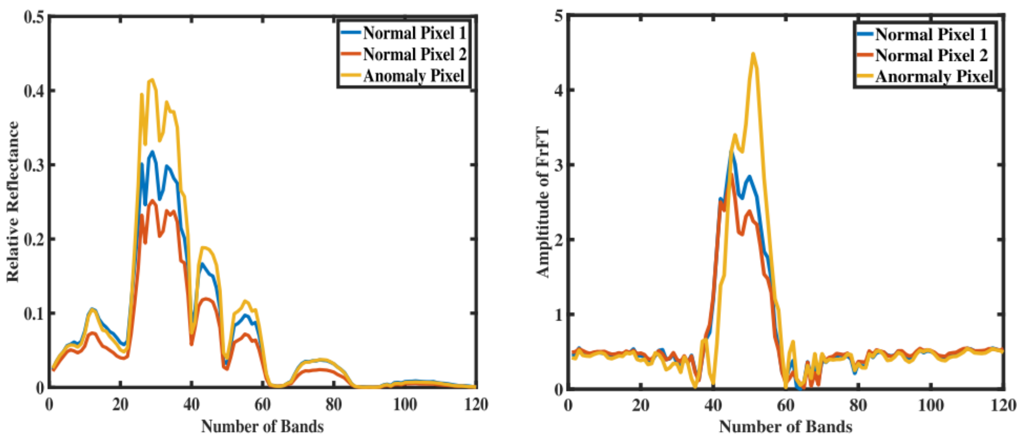


Figure 3.16: Two different domain representations of background and anomaly pixels. The right graph shows spectral domain, while the left mixed spectral and Fourier transform domain [31].

3.4.8.4 CRD

CRD is based on the fact that background pixels often can be represented as a weighted sum of its neighbour pixels, while an anomaly cannot. This algorithms detect anomalies by looking at the difference between the weighted neighbour pixels in a window consisting of an inner and outer frame, and the current pixel in the middle of the inner frame [32].

3.4.8.5 F-MGD

The F-MGD algorithm uses morphological reconstruction and guided filter to detect anomalies. Morphological filtering can be used as a tool to extract region shapes of the image [33].

3.4.9 Classification

Classification algorithms segregate the spatial image into different groups, where each group contains elements that have similar characteristics. Figure 3.17 illustrates classification. The pixels are marked with a color that indicates what class the pixels belong to.

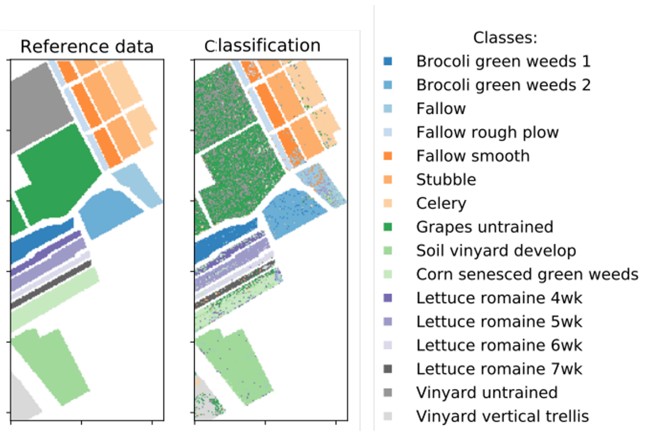


Figure 3.17: Two images, where image one is reference scene showing the ideal classification, while scene two shows the same scene classified by a classification algorithm [34].

Even though there are a lot of classification methods, this design space exploration only includes SVM classification.

3.4.9.1 SVM

SVMs are initially a binary classification algorithm. They classify data according to a hyperplane, so pixels on one side are classified as one type of pixels, while the pixels on the other side of the hyperplane are classified as the other data class [35] [36].

The support vector machine has support vectors that are data points with known classifications. The hyperplane is the plan that best separates these support vectors of the different classes.

Figure 3.18 illustrates the concept of SVM classification in two dimensions, where the fully red pixels are support vectors of a red class, and the fully blue pixel is a support vector of the blue class. The other data points in figure 3.18 are pixels that are classified according to the hyperplane [36].

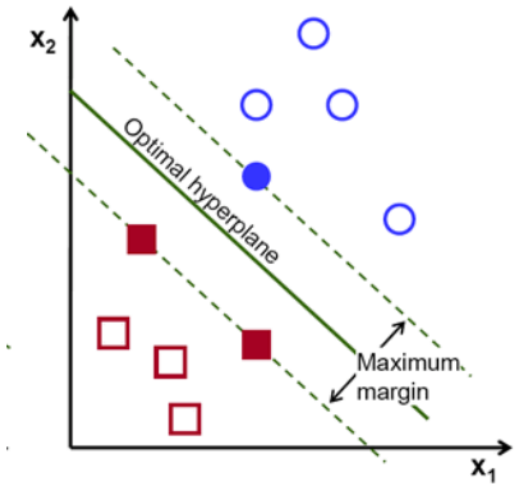


Figure 3.18: SVM classification of data points given the support vectors fully colored in red and blue, and the hyperplane splitting dividing the vector space into two areas [36].

SVM can also be used for multi class problems. The multi class problem consists of several binary classifications, and based on the classification results, we can determine in what class the data points belongs [35].

3.4.10 CCSDS123 Compression

There are a lot of algorithms that compress images. Algorithms labeled as compression algorithms are often algorithms that can represent the images in a more compact but less readable manner and decompress the image back to its original shape and size. CCSDS123 algorithms are such compression algorithms. This thesis includes the lossless CCSDS123-B1 algorithm and the near lossless CCSDS123-B2 algorithm in our design space exploration.

Chapter 4

Implementation

4.1 Models

In this design space exploration, we use two models to evaluate the onboard data processing pipelines.

4.1.1 MAUT Model

The first model is based on the MAUT methodology, which has one metric for the design's overall benefit and one for the associated costs. The MAUT model's combined benefit is a combination of the processing pipelines' data accuracy and outputted data size. The higher the data quality and lower the outputted data size, the higher the overall benefit. The cost metric is the processing time; the higher the processing time, the higher the cost.

The MAU score is computed by the attributes' SUV scores and associated weights. In this design space exploration, we weigh accuracy more heavily than outputted data size because the data quality is considered more important than outputted data size. Nevertheless, the outputted data size is important too because it can limit the data throughput of the system. Consequently, in this design space exploration, the data accuracy is weighted to 0.6, and the outputted data size to 0.4. Equation (4.1) describe the MAU score mathematically.

$$MAU = 0.6 \cdot SVU_{\text{acc}}(\widehat{x}_{\text{acc}}) + 0.4 \cdot SVU_{\text{size}}(\widehat{x}_{\text{size}}) \tag{4.1}$$

4.1.1.1 $SVU_{\mathbf{acc}}$

In this design space exploration, we use the SVU graph shown in figure 4.1 to convert estimated data accuracy to a SVU_{acc} score.

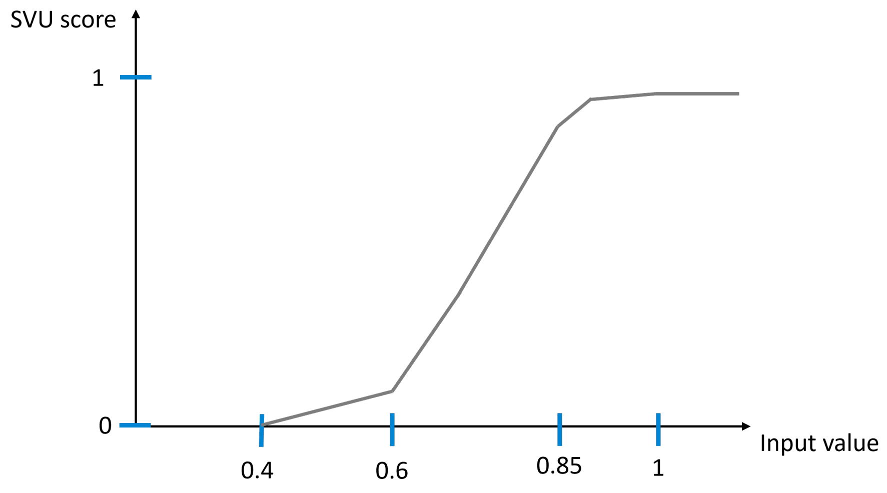


Figure 4.1: The SVU graph for the accuracy attribute.

The SVU graph shown in figure 4.1 allows the accuracy to sink to 85% without too much reduction in benefit. When the estimated accuracy is less than 85% the SVU value falls substantially as data with an accuracy below 85% is not much reliable.

4.1.1.2 SVU_{size}

The graph shown in figure 4.2 transform the estimated outputted data size to a SVU_{size} score. In contrast to the fixed accurate SVU graph, the SVU graph for outputted data size depends on the size of the hyperspectral image. A two dimensional representation of an image is weighted equally good for images of high spatial sizes, as for those with small spatial sizes. The SVU score is only dependent on how many spectral bands used to express the outputted data.

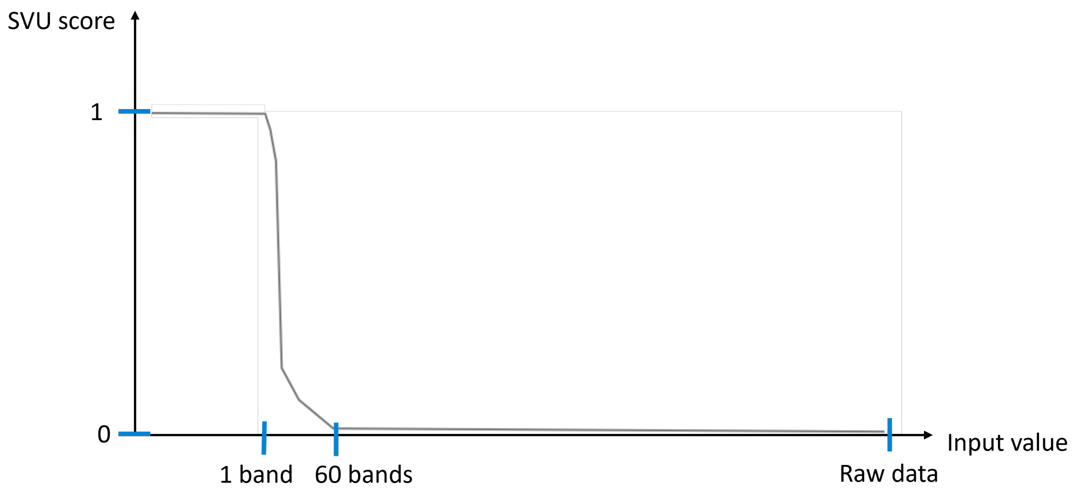


Figure 4.2: The SVU graph for the outputted data size attribute.

As seen in figure 4.2 the best possible data output size is a 2D representation of the hyperspectral image. The SVU score reduces fast as the output size increase, and the worst output size is the size of the raw hyperspectral cube.

Hyperspectral cubes typically consist of more than 60 bands, but representing the data with 60 bands is still much more data than representing it with only one band. The SVU score will therefore almost reach zero when the outputted data consist of 60 bands.

4.1.2 Second Model

The second model includes the outputted data size in the cost metric rather than in the benefit metrics. Data size multiplied by downloading rate produces a time measurement that can be added to the processing time estimate. Together, the two time estimates indicate the total time usage required for handling the hyperspectral data. This model simply compare estimated data accuracy with estimated total time usage.

The optimal processing pipelines can capture, process, and download hyperspectral data in one orbital pass. It is only possible to download data when the satellite passes a ground station, in other words, there is limited time to download data. It is also limited time for processing the hyperspectral data before needing to download it. The time constraints of an optimal pass can be summarized as follows,

- The data processing should provide the best possible accuracy.
- The outputted data size should be so small that the data can be downloaded within one orbital pass.
- The processing chain's run time should be so small that image capturing, data processing, and downloading completes within one orbital pass.

By applying the listed constraints to the second model we can easily check whether the onboard data processing pipelines are optimal or not.

Equation 4.2 shows mathematically how the cost is a function of estimated run time and data downloading time.

$$\text{cost} = \text{estimated processing time} + \text{estimated data size output} \cdot \text{estimated data downloading rate} \tag{4.2}$$

4.2 Estimations

4.2.1 Outputted Data Size Estimations

As explained in the background section, we can estimate outputted data size by understanding the objective of the algorithms. The algorithms in this design space exploration can be divided into three groups based on their objectives.

1. Algorithms that first and foremost aim to compress the data.
2. Algorithms that intend to correct the captured data making it resemble the ground truth to a higher degree.
3. Algorithms that draw conclusions about the data.

By drawing conclusions about the data, we mean that by analyzing the inputted data, the algorithms determine facts about the data. Examples of such facts are the number of targets or anomalies in the pictured area.

This group can reduce the data size significantly, just like the compression algorithms. The difference between this interpreting group of algorithms and the compression algorithms is that the compression algorithm represents the same data but with fewer data points, while the interpreting algorithms output metadata of inputted data.

4.2.1.1 Outputted Data Size Estimates for Compression Algorithms

The algorithms in this design space exploration that falls into the compression algorithm group are spectral and spatial binning, dimensionality reduction, lossless CCSDS123-B1 compression, and near lossless CCSDS123-B2 compression.

4.2.1.1.1 Binning

Given a spectral binning factor, spectral binning lowers the number of spectral pixels to the number of spectral binning groups, as follows

$$\text{Spectral binning groups } = \left\lceil \frac{\text{spectral pixels}}{\text{binning factor}} \right\rceil. \tag{4.3}$$

Given a spatial binning factor, spatial binning does the same as spectral binning but reduces the spatial pixels,

$$\text{Spatial binning groups} = \left\lceil \frac{\text{spatial pixels}}{\text{binning factor}} \right\rceil. \tag{4.4}$$

Spatial binning either reduces the number of frames or the number of spatial pixels within a frame.

4.2.1.1.2 Dimensionality Reduction

Dimensionality reduction reduces the number of components in the spectral dimension to a predefined number of components.

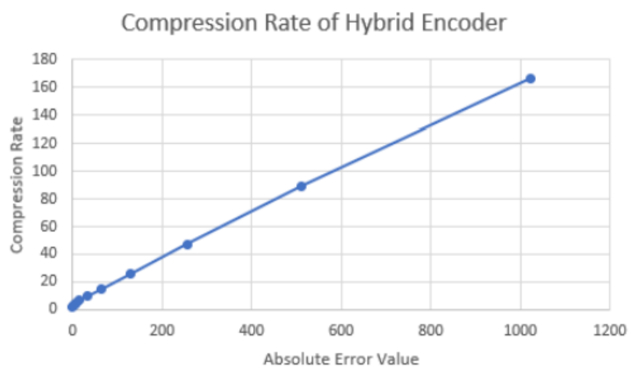
4.2.1.1.3 CCSDS123 Compression

In contrast to binning and dimensionality reduction, the CCSDS123-B1 and CCSDS123-B2 algorithms do not have a deterministic compression factor. Therefore, it is not enough to understand the algorithms to conclude how much reduction they provide. Another way to estimate the outputted data size is to look into performance studies. According to Grøtte et. al CCSDS123-B1 has a reduction factor of 2.25 [3].

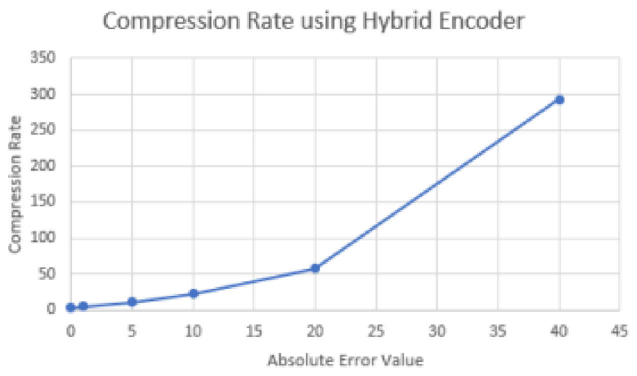
Boothby shows how the data reduction correlates with the outputted data quality in his master thesis, where he define the reduction rate as [37],

$$\text{Compression rate } = \frac{\text{Original size}}{\text{Compressed size}}. \tag{4.5}$$

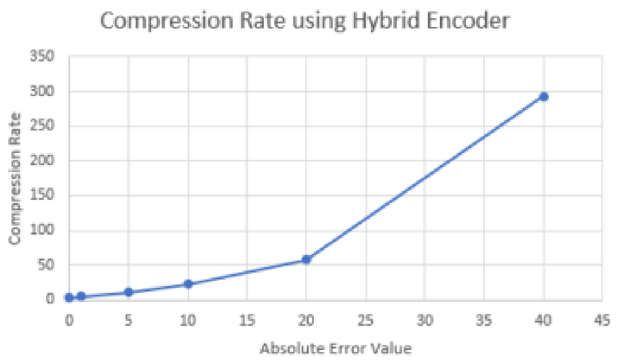
Figure 4.3a, 4.3b and 4.3c are fetched from Boothby's master thesis, and display the data reduction rate as a function of absolute error for the three different images shown in figure 4.4a, 4.4b and 4.4c, respectively.



(a) Compression rate for a the low entropy image shown in figure 4.4a.

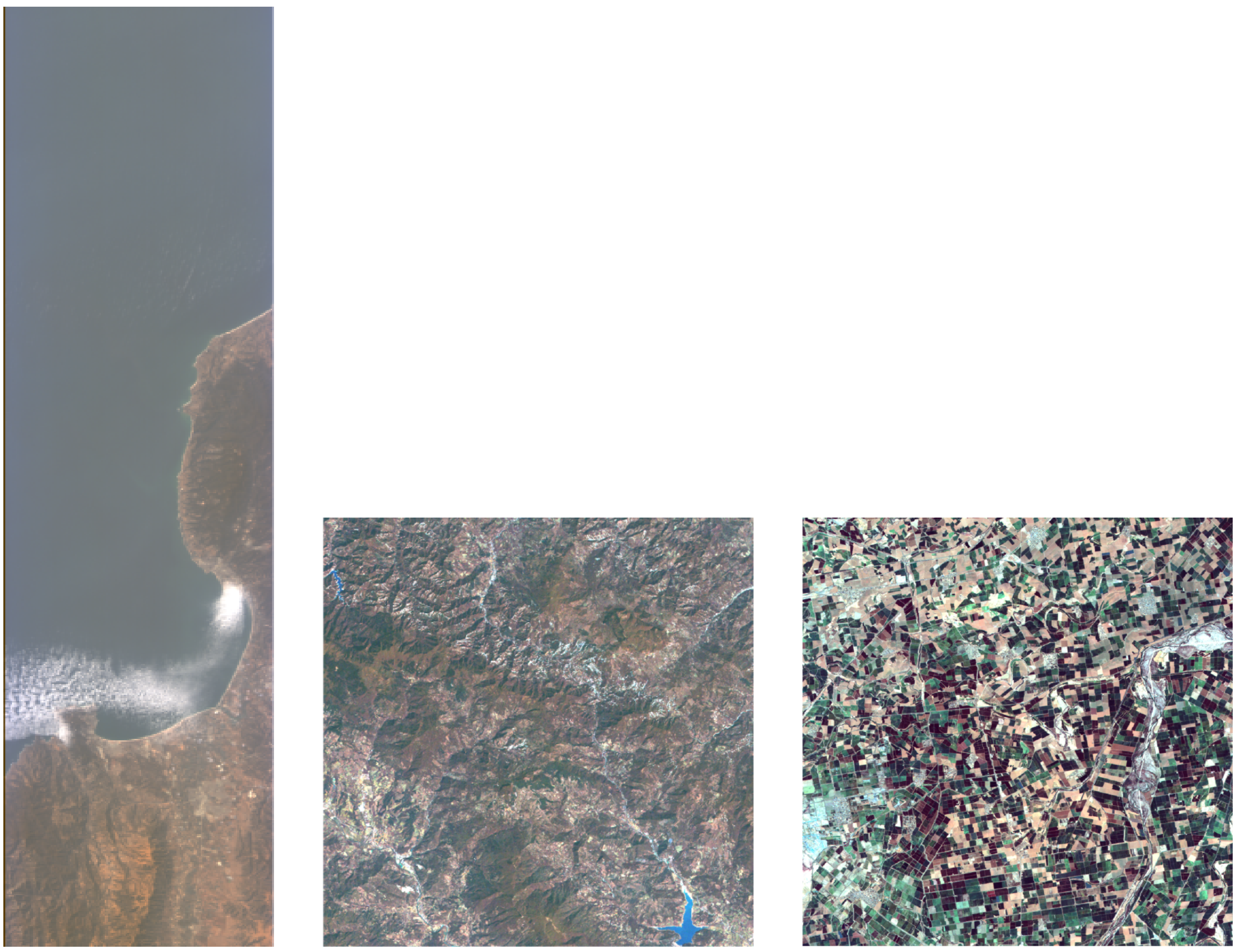


(b) Compression rate for a the high entropy image shown in figure 4.4b.



(c) Compression rate for a the high entropy image shown in figure 4.4c.

Figure 4.3: The CCSDS123-B2 compression rate as a function of absolute error for three different images [37].



(a) A low entropy image. **(b)** A high entropy image. **(c)** A high entropy image.

Figure 4.4: Examples of one low entropy image, and two high entropy images [37].

By absolute error, Boothby refers to an error parameter in the CCSDS123-B2 algorithm that can take values in the range 0 to $2^{16}-1$.

As we can see in the graphs the entropy level affects the compression rate. Therefore, the information regarding figure 4.4a is the most relevant to the HYPSO project because the figure has low entropy, and HYPSO aims to take images of the ocean which is likely to have a low entropy considering the ocean's limited environmental variations.

As shown in figure 4.3a, there is almost an linear relationship between compression rate and absolute error value. Consequently we use this near linear relationship to estimate data reduction in this design space exploration.

4.2.1.2 Outputted Data Size Estimates for Correcting Algorithms

In this design space exploration, the correcting algorithms are bad pixel detection and correction, smile and keystone correction, radiometric calibration, and geometric registration. Except for radiometric calibration, neither of these algorithms affect the input data size. Radiometric calibration can increase the image resolution and thereby increase the image size. In this design space exploration, we use radiometric calibration without increasing the image resolution, and none of the correcting algorithms will therefore affect the outputted data size.

4.2.1.3 Outputted Data Size Estimates for Interpreting Algorithms

The interpreting algorithms in this design space exploration are target detection, anomaly detection, and classification. The outputs of target detection and anomaly detection are a two dimensional image that tells how likely it is that the spatial pixel holds a target or an anomaly. In other words, the third dimension no longer contains spectral information but a probability map. The probability map can be expressed by what constitutes one spectral band.

The classification algorithm does not show a two dimensional probability map like target detection and anomaly detection but divides the spatial image into different categories. As long as there are fewer categories than the resolution of a spectral value, we can express the classes through the resolution of only one spectral band. In this design space exploration we assume that described case for classification.

To summarize, the outputted data size of the interpreting algorithms is a two dimensional image that uses what constitutes one spectral value to represent the interpreting information.

4.2.1.4 Overview of the Data Output Estimations

Table 4.1 display an overview of how the processing modules in this design space exploration affect an inputted hyperspectral data cube of dimensions *frames* x *framesamples* x *bands*.

Table 4.1: An overview of outputted data size for algorithms used in this design space exploration given input data size of *frames* x *framesamples* x *bands*

Algorithm Names	**Outputted Data Size**
Spectral binning	$\textit{frames} \text{ x } \textit{framesamples} \text{ x } \left\lceil \frac{\textit{bands}}{\textit{binning factor}} \right\rceil$
Spatial Binning (framesamples)	$\textit{frames} \text{ x } \left\lceil \frac{\textit{framesamples}}{\textit{binning factor}} \right\rceil \text{ x } \textit{bands}$
Spatial Binning (frames)	$\left\lceil \frac{\textit{frames}}{\textit{binning factor}} \right\rceil \text{ x } \textit{framesamples} \text{ x } \textit{bands}$
PCA, MNF, ICA	*frames* x *framesamples* x *dimensionality reduced bands*
CCSDS123-B1	(*frames* x *framesamples* x *bands*)/2.25
CCSDS123-B1	(*frames* x *framesamples* x *bands*)/CR, der CR = CR(absolute error) shown in figure 4.3a
Bad Pixel Detection and Correction, Smile and Keystone Correction, Radiometric Calibration, Geometric Registration	*frames* x *framesamples* x *bands*
Target Detection, Anomaly Detectiom, Classification	*frames* x *framesamples* x 1

4.2.2 Data Accuracy Estimate

The accuracy estimates are based on previous performance studies and assumptions on how algorithms affect each other. Even though it is preferable to estimate data accuracy based on only one quality metric, this design space exploration estimates accuracy based on several metricizes. These metricizes are

- AUC scores,
- Accuracy scores,
- Signal to noise ratio,
- Resolution decreasing,
- Data retainment, and
- Visual inspection of images.

The total accuracy estimate of the processing pipeline is computed by independently looking at the accuracy metrics of the algorithms and multiplying them.

In this design space exploration, we assume a raw cube has a data accuracy of 84% and that algorithms can increase or decreases the quality. By assuming that the data accuracy is only 84%, there are some room for increasing the data quality.

4.2.2.1 Estimation of Data Accuracy to the Correcting Algorithms

Correcting algorithms increases the quality of the hyperspectral data. As mentioned in section 4.2.1, the correcting algorithms are bad pixel detection and correction, smile and keystone correction, radiometric calibration, and geometric registration. These algorithms make the data resemble reality to a higher degree, thereby increasing quality. Nevertheless, quantifying how much these algorithms improve the quality is not trivial.

The smile and keystone and radiometric calibration study shows that wavelength errors and dark current effects lessen when applying the algorithms [24]. The Geometric registration study illustrates how geometric correction changes the image proportions to resemble the earth layout to a higher degree [29]. The bad pixel detection and correction study demonstrate that bad pixel detection algorithms can detect and correct almost all bad pixels [22]. Even though the detection and correction rate are different for different bad pixel detection and correction algorithms, we simply assume an accuracy increase of 5% for them all.

Even though the mentioned algorithms improve the quality of the hyperspectral data, it is hard to know by how much. Consequently, we simply assume that applying one of the correcting algorithms increases the image quality by 5%. Nevertheless, knowing that the target detection algorithms and the classification algorithm only rely on spectral information, we can assume that geometric registration will not affect the result of these algorithms. On the other hand, anomaly detection is not as sensitive to particular spectral signatures as target detection and classification. It would probably detect anomalies even though the data suffers from dark current or spectral inaccuracy. Therefore, we assume smile and keystone correction and radiometric calibration do not increase data quality when using anomaly detection algorithms. The following list summarizes the assumed accuracy dependencies in this design space exploration.

- If the processing pipeline includes target detection or classification algorithms, geometric registration will not increase the data quality, but if it does not, it improves the quality by 5%.
- If the processing pipeline includes anomaly detection, smile and keystone correction and radiometric calibration will not increase the data quality, but if it does not, they boost it by 5% each.

4.2.2.2 Estimation of Data Accuracy to the Compressing Algorithms

The compression algorithms can increase, decrease or not affect the data quality.

4.2.2.2.1 Binning

The theoretical spectral bandpass for a hyperspectral camera tells how a spectral wavelength will disperse across the spectral values. The theoretical spectral bandpass for the HYPSO-1 camera is 3.33 nm, which covers nine spectral pixels [38]. The HYPSO-1 mission can bin up to 9 spectral values without losing spectral resolution [3]. In this design space exploration, we will assume the use of the same hyperspectral camera as in HYPSO-1, and therefore also the same theoretical spectral bandpass.

By having a theoretical bandpass on nine spectral values, it is possible to bin spectral pixels by x and lose resolution corresponding to binning x/9 spectral pixels. The spatial resolution does not have the same bandpass features as spectral resolution. Therefore, when binning spatial pixels by x, the resolution decreases by x. Consequently, spectral binning is less lossy than spatial binning.

For the HYPSO-1 mission, a spatial and spectral resolution decrease of two corresponds to doing spatial binning of two and a spectral binning of eighteen values.

In general, we estimate the data accuracy as a function of resolution decrease, as shown in figure 4.5.

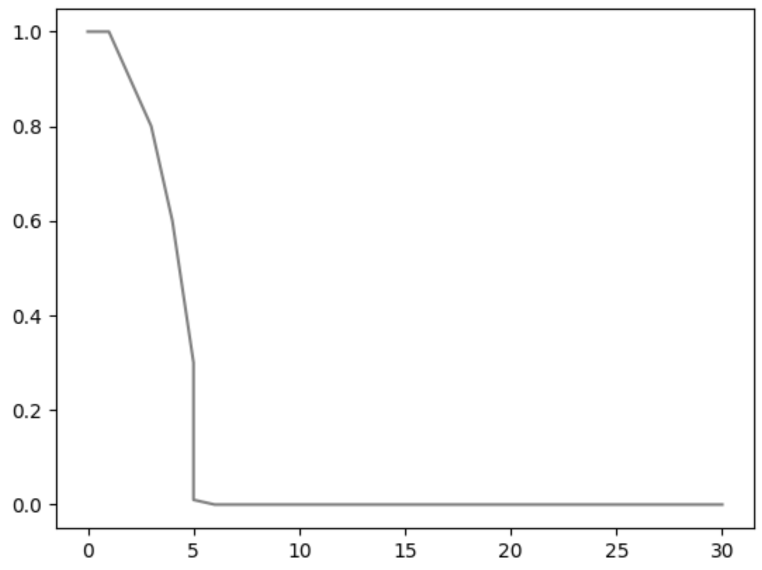


Figure 4.5: Data accuracy as a function of resolution decrease. X axis shows resolution decrease factor, and y axis the data accuracy.

The x axis displays how much the resolution decreases, and the y axis the estimated accuracy. Figure 4.5 shows that the data accuracy decreases more rapidly as the resolution decreases, and when reached a resolution decrease by five, the estimated accuracy becomes zero.

In addition to decreasing resolution, binning will also increase the SNR [3] [39]. Figure 4.6 shows how the SNR is affected when binning pixels in a frame [39]. In this design space exploration, we will use the displayed relationship to estimate how binning affect the SNR, for both spectral and spatial binning.

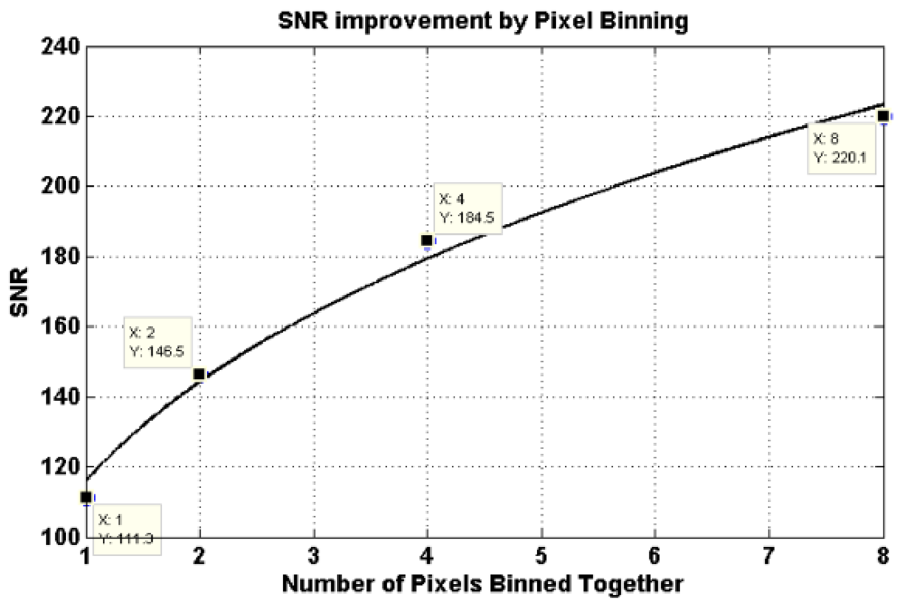


Figure 4.6: SNR as a function of binned pixels. The graph is a curve fit of the samples shown in the figure [39]

To estimate a data accuracy based on the increase of SNR, and the decreasing resolution we use the mathematical expression,

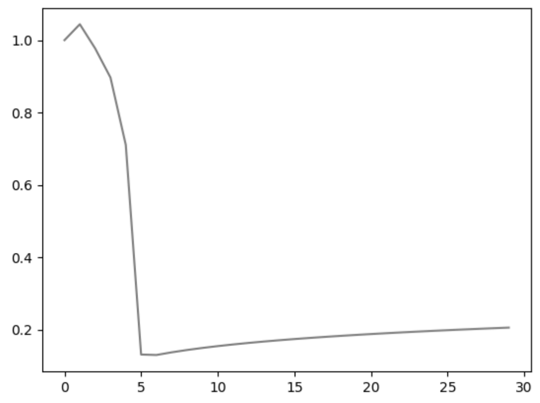
$$\text{Data accuracy(resolution decrease)} = 0.004 \cdot \text{ SNR(binning factor)}, \tag{4.6}$$

where the resolution decrease is a function of binning factor, and *SNR(binning factor)* is the relationship displayed in figure 4.6. Equations (4.7) and (4.8) show the resolution decrease as a function of binning factors for spatial binning and spectral binning respectively.

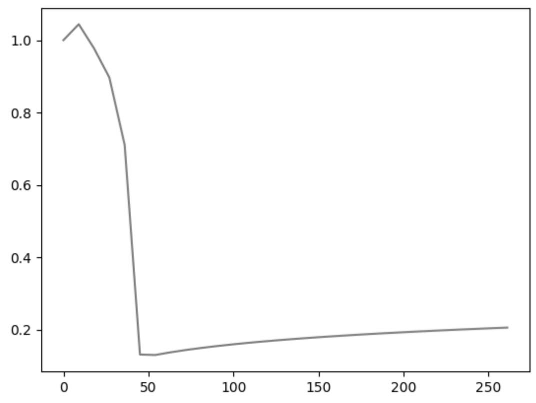
$$\text{Spatial resolution decrease } = \text{ spatial binning factor} \tag{4.7}$$

$$\text{Spectral resolution decrease } = \frac{\text{spectral binning factor}}{9} \tag{4.8}$$

Figures 4.7a and 4.7b show the estimated data accuracy on the y axis as a function of spatial and spectral binning factors (x axis), respectively.



(a) Data accuracy estimate for spatial binning given a spatial binning factor.



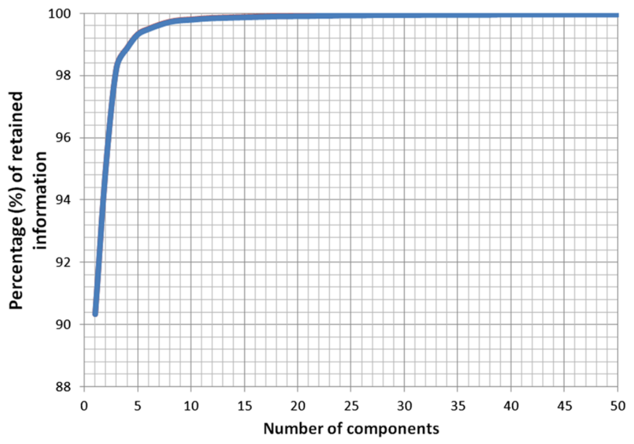
(b) Data accuracy estimate for spectral binning given a spectral binning factor.

Figure 4.7: The relationships of data accuracy and spatial and spectral binning. The x axses shows the binning factor, and the y axses shows the estimated data accuracy given a binning factor.

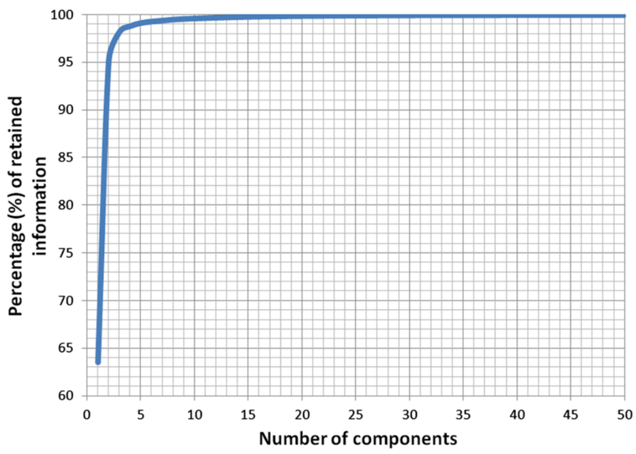
The estimated data accuracy curves are similar for spatial and spectral binning. However, the scale of the binning factors differs substantially. A spectral binning of 35 values gives the same accuracy estimate as the spatial binning of 5 values.

4.2.2.2.2 Dimensionality Reduction

The amount of retained data after dimensionality reduction determines the data accuracy estimation of the dimensionality reduction algorithms. Figure 4.8a and 4.8b are fetched from the performance study "FPGA implementation of the principal component analysis algorithm for dimensionality reduction of hyperspectral images" and show how much of the original data the PCA retains when representing the data with different amounts of principal components [40]. In one case, 99.9% of the original data is contained within 16 principal components, while the other case requires about 24 components. In this design space exploration, we use an approximate average of the two data quality measurements to determine data quality estimations of PCA.



(a) A graph displaying the retained information as a function of components when applying PCA on a hyperspectral image [40].



(b) Another graph displaying the retained information as a function of principal when applying PCA on another hyperspectral image [40].

The author has not found a similar performance study for the other dimensionality reduction algorithms, ICA and MNF. However, the study "Evaluation of dimensionality reduction techniques in hyperspectral imagery and their application for the classification of terrestrial ecosystems", shows the output of a classification algorithm when preprocessing with PCA, ICA, and MNF [25]. Figure 4.9 shows the accuracy result of the classification when converting the spectral dimension to a reduced domain.

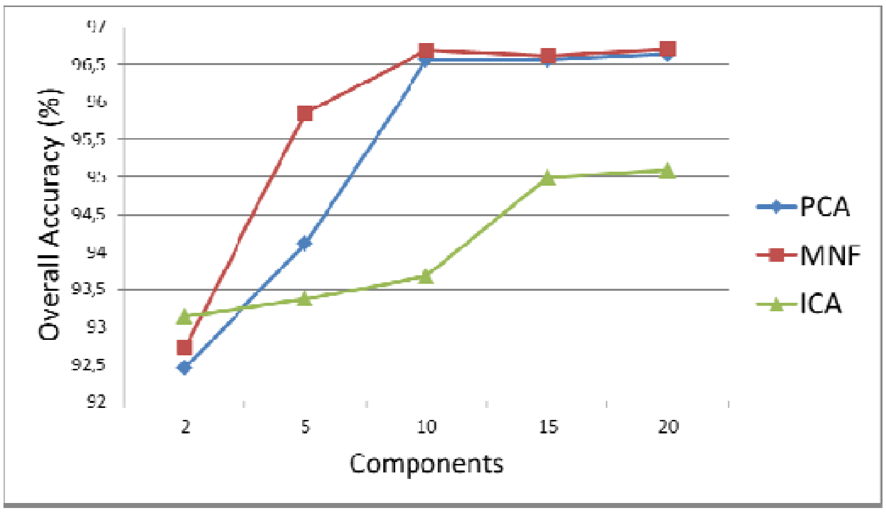


Figure 4.9: The SVU graph for the accuracy attribute.

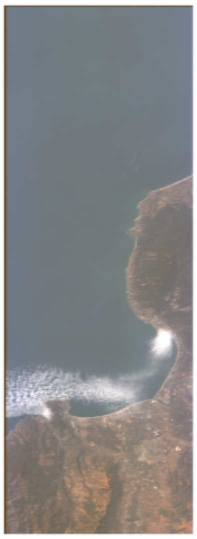
Figure 4.9 shows that PCA and MNF affect the classification output similarly, while ICA reduces the quality slightly. Consequently, we estimate that the accuracy estimate of MNF is the same as for PCA, but 0.984 times smaller accuracy for ICA compared to the PCA estimate. We reduce the accuracy 0.984 times as figure 4.9 shows that ICA Saturate at approximately 0.984 times lower score than PCA and MNF.

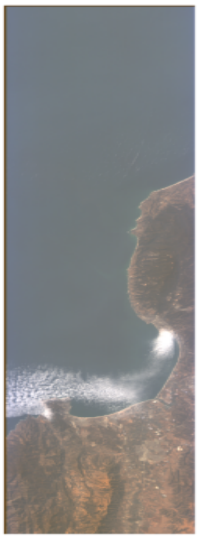
4.2.2.2.3 CCSDS123 Compression

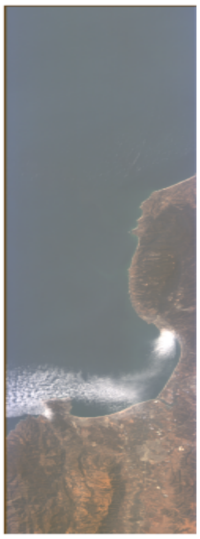
The lossless CCSDS123-B1 compression algorithm will not affect the data accuracy.

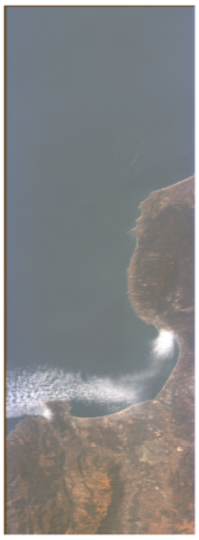
The near lossless compression algorithm CCSDS123-B2, on the other hand will. Figure 4.3a in Section 4.2.1.1.3 illustrates how data compression rate is correlated with absolute error value. In Boothby implementation of the CCSDS123-B2 algorithm, the absolute error value is an parameter that determines the compression rate and decrease of data quality. Exactly how the absolute error affects the overall data quality is not elaborated in Boothby's thesis. Nevertheless, he claims that an absolute error in the range of 0 to 128 does not give a severe image degradation for the image shown in figure 4.4a [37].

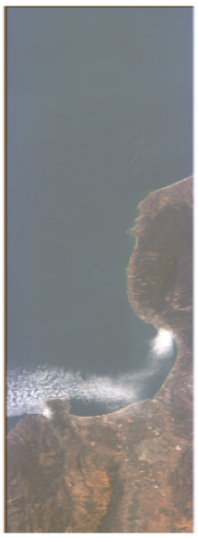
In this design space exploration, we estimate CCSDS123-B2's data accuracy based on Boothby's claim that absolute error beneath 128 is not too harmful, and the visual inspection of what happens with the data quality when applying different absolute error values. The subfigures of figure 4.10 are fetched from Boothby's master thesis and used for the visual inspection of data quality versus absolute error value of the image shown in figure 4.4a.

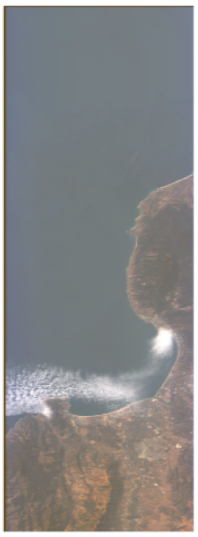

(a) An absolute error of 2

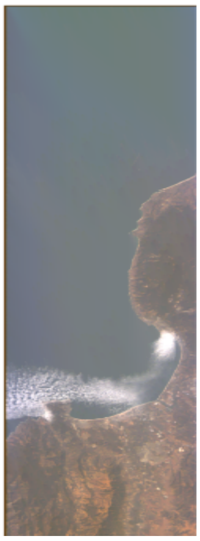

(b) An absolute error of 2

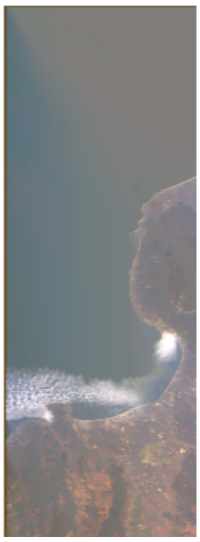

(c) An absolute error of 8

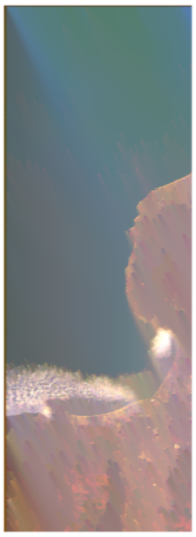

(d) An absolute error of 16


(e) An absolute error of 32


(f) An absolute error of 64


(g) An absolute error of 128


(h) An absolute error of 256


(i) An absolute error of 512

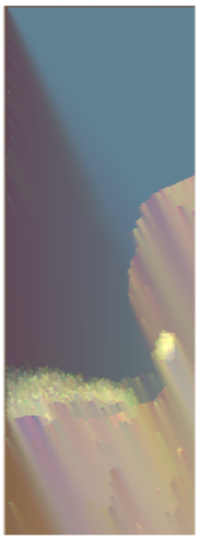

(j) An absolute error of 1024

Figure 4.10: Figures showing how different absolute error values affect the image quality of figure 4.4a [37].

Boothby states that an absolute error value of 128 will not affect the data quality too much, we therefore estimate an data accuracy of 85% when there is an absolute error value of 128 [37]. As shown in figure 4.10h and 4.10j, the data quality turns quite bad when the absolute error value is 256, and very bad when it is 1024. Consequently we let the image accuracy rapidly decrease in the interval between 128 to 1024. If the absolute error value is 1024 or higher, we estimate the data quality to be zero. Figure 4.11 displays this design space exploration's accuracy estimate as a function of absolute error.

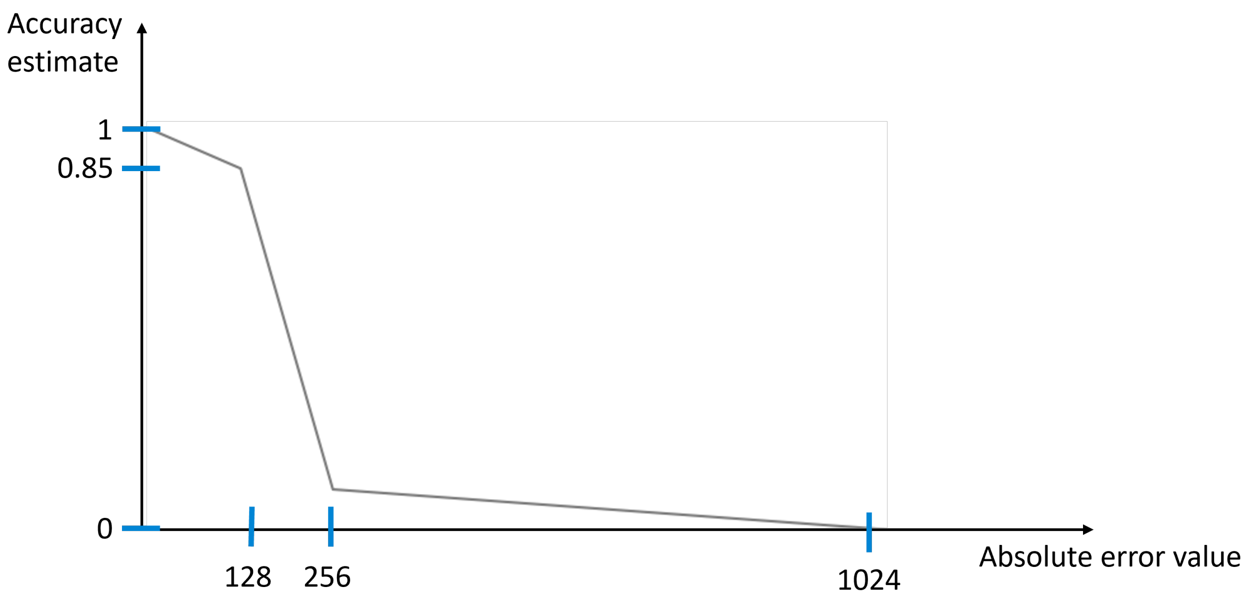


Figure 4.11: Estimated data accuracy as an function of absolute error value.

4.2.2.3 Estimation of Data Accuracy to the Interpreting Algorithms

Table 4.2 lists the recorded AUC scores for the target detection and anomaly detection and accuracy score for the classification algorithm. In this design space exploration we use these AUC and accuracy scores to estimate the data accuracy. The table also list the performance studies where the AUC and accuracy scores are collected. Some of the performance studies record the algorithms AUC score for several images. When the performance studies record several AUC scores table 4.2 displays them as the average score.

Table 4.2: AUC and accuracy scores for target and anomaly detection, and classification algorithms, and from what performance study the AUC and accuracy scores are collected.

Algorithm	AUC/Accuracy	Performance Study
SAM	0.889 (AUC)	[41]
CEM	0.9982 (AUC)	[42]
ACE	0.9976 (AUC)	[42]
ASMF	0.9982 (AUC)	[42]
GRX	0.9420 (AUC)	[43]
LRX	0.9604 (AUC)	[43]
CRD	0.9673 (AUC)	[43]
FrFT-RX	0.9657 (AUC)	[44]
FMGD	0.9947 (AUC)	[33]
SVM	0.966 (Accuracy)	[25]

4.2.3 Run Time Estimation

In this subsection, we explain how to do the run time estimations.

For each processing algorithm, the following subsections provide an operation count. The operation counts multiplied by an operation frequency are the run time estimates. HYPSO-1 has an operating frequency of 667MHz, and we will assume the same when making run time estimates.

The utility section explains the operation counts of the algorithms used in this design space exploration. The run time of the algorithms can be dependent on certain variables. If nothing else is stated, assume that these variables have been given their typical values.

4.2.3.1 Utility

4.2.3.1.1 Dot Product

Equation (4.9) shows how the dot product of vectors A and B with size N are obtained by CPUs.

$$\begin{gathered} A \cdot B = a_0 \cdot b_0 + a_1 \cdot b_1 + \ldots + a_n \cdot b_n \\ \text{operations } = N \cdot \text{ multiplications } + (N-1) \cdot \text{ additions} \end{gathered} \tag{4.9}$$

Figure 4.12 shows an suggested FPGA implementation of dot product [45].

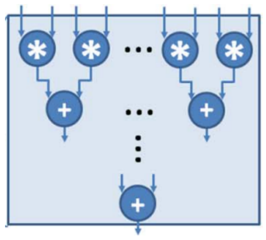


Figure 4.12: Purposed hardware circuit for dot product [45]

The number of operations are,

$$\text{operations } = N \cdot \text{ multiplications } + \lceil log(N) \rceil \cdot \text{ additions} . \tag{4.10}$$

4.2.3.1.2 Matrix multiplication, Matrix Vector Multiplication, and Sample Correlation Matrix

Figure 4.13 illustrate matrix multiplication of the matrixes A of size NxM=4x2 and B of size MxL=2x3.

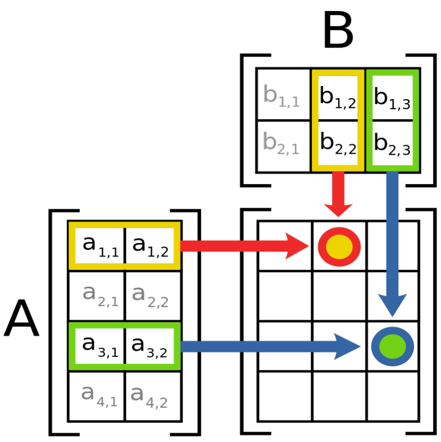


Figure 4.13: Concept of matrix multiplication [46].

As we can see in figure 4.13, that the operations needed in matrix multiplication is

$$\text{operations } = N \cdot L \cdot \text{dot products of M elements} \tag{4.11}$$

The hardware implementation of matrix mulitplication can also be obtained by series of dot productions. However, we can add several dot product processing hardware to operate in parallell. The number of operations is therefore,

$$\text{operations } = N \cdot L \cdot \frac{1}{K} + \text{dot products of M elements,} \tag{4.12}$$

where K is the number of dot product hardware circuits in the matrix multiplication hardware.

If B is a vector instead of a matrix, the we use the same procedure as described for matrix matrix multiplications, but let L be 1.

The sample correlation matrix of a Matrix A is given in (4.13) [47].

$$\mathbf{R} = \frac{1}{N}\mathbf{A}\mathbf{A}^T \tag{4.13}$$

The sample correlation matrix is a matrix multiplication of A and its transpoase divided by the row count of the matrix, N, and the operation counts is therefore,

$$\text{operations } = 1 \cdot \text{ matrix multiplication } + M^2 \cdot \text{ divisions} \tag{4.14}$$

It is the same operation count for hardware generation of the correlation matrix (but using hardware circuits as described above).

To update the sample correlation matrix by a sample **a** of size $Mx1$ we do the following,

$$\mathbf{R}_{\text{updated}} = \mathbf{R} + \mathbf{a}\mathbf{a}^T, \tag{4.15}$$

where the generation of $\mathbf{a}\mathbf{a}^T$ requires M^2 multiplications. The total number of operations is therefore,

$$\text{operations } = M^2 \cdot (1 \cdot \text{multiplications} + 1 \cdot \text{ additions }) \tag{4.16}$$

4.2.3.1.3 Distance

$$\begin{aligned} ||A|| &= \sqrt{a_0^2 + a_1^2 + ...a_n^2} \\ \text{operations } &= n \cdot \text{multiplication} + (n-1) \cdot \text{subtraction} + \text{sqrt} \end{aligned} \tag{4.17}$$

4.2.3.1.4 Diagonal Matrix Multiplication

Equation (4.18) shows the multiplication of an arbitary matrix A with an diagonal matrix K both of size NxN.

$$\begin{bmatrix} k_1 & 0 & ... & 0 \\ 0 & k_2 & ... & 0 \\ ... & ... & ... & ... \\ 0 & 0 & ... & k_N \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & ... & a_{1N} \\ a_{21} & a_{22} & ... & a_{2N} \\ ... & ... & ... & ... \\ a_{N1} & a_{N2} & ... & a_{NN} \end{bmatrix} = \begin{bmatrix} k_1a_{11} & k_1a_{12} & ... & k_1a_{1N} \\ k_2a_{21} & k_2a_{22} & ... & k_2a_{2N} \\ ... & ... & ... & ... \\ k_Na_{N1} & k_Na_{N2} & ... & k_Na_{NN} \end{bmatrix} \tag{4.18}$$

As we can see, the elements of the resulting matrix equals the elements of the matrix A multiplied by a factor k_i. The number of operations are therefor,

$$\text{operations } = N \cdot N \cdot \text{multiplications.} \tag{4.19}$$

4.2.3.1.5 Mean

The mean of a N elements is,

$$\bar{x} = \frac{1}{N} \sum_{i=0}^{N-1} x_i, \tag{4.20}$$

which means a total operation count of,

$$\text{operations } = N \cdot \text{ additions } + 1 \cdot \text{ division} \tag{4.21}$$

Updating an mean of a sliding window is adding a scaled new pixel and subtraction the scaled oldest one. The number of operations needed for this is,

$$\text{operations } = 1 \cdot \text{ addition } + 1 \cdot \text{ subtraction } + 2 \cdot \text{ divisions} \tag{4.22}$$

4.2.3.1.6 Variance

Equation (4.23) shows the variance of a N elements is,

$$s = \frac{1}{N-1} \sum_{i=0}^{N-1} (x_i - \bar{x})^2, \tag{4.23}$$

which needs the following number of operations,

$$\text{operations } = N \cdot (1 \cdot \text{ additions } + 1 \cdot \text{ subtraction } + 1 \cdot \text{ multiplication }) + \text{ division .} \tag{4.24}$$

4.2.3.1.7 Sorting

Gonzalez et. al. purpose an sorting circuit consisting of a chain of replicated hardware. figure 4.14 illustrates one of these hardware cells [48].

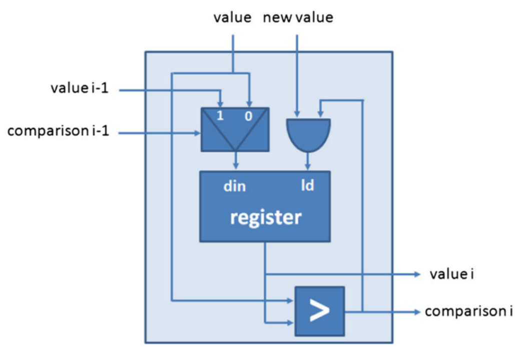


Figure 4.14: One hardware block in a sorting circuit consisting of several of these hardware blocks [48].

The sorting hardware uses one clock cycles to sort one sample, so to sort N samples the circuit uses N clock cycles.

Merge sort is a software algorithm for sorting, which has complexity of $O(n \log n$ [49]. The concept of the sorting algorithm is to divide the elements into subgroups, and compare the elements of the subgroups [50]. We therefore estimate the operation count of merge sort to be,

$$\text{operations} \; = n \cdot \; \log n \cdot (1 \cot \; \text{divide} \; + 1 \cdot \; \text{compare} \;) \tag{4.25}$$

4.2.3.1.8 Median

The median is the midle element in a sorted list. The number of needed operations is therefore,

$$\text{operations} \; = \; \text{sort list} \; + 1 \cdot \; \text{division} \;, \tag{4.26}$$

where the division is the list size divided by two, which gives us the median element.

4.2.3.1.9 Inverse Matrix Update

The Sherman morris algorithm update the inverse of a matrix when adding more samples new samples [18]. When using sliding windows we add a new sample and remove the oldest sample, as shown in (**??**).

$$\mathbf{S_i} = \mathbf{S_{i-1}} + \left(\mathbf{x_{new}x_{new}^T} - \mathbf{x_{old}x_{old}^T}\right) \tag{4.27}$$

The Sherman morris algorithm do this according to the following procedure [18].

$$\mathbf{T}^{-1} = \mathbf{S}_{i-1}^{-1} - \frac{\mathbf{S}_{i-1}^{-1}\mathbf{x}_{new}\mathbf{x}_{new}^{T}\mathbf{S}_{i-1}^{-1}}{\mathbf{x}_{new}^{T}\mathbf{S}_{i-1}^{-1}\mathbf{x}_{new} + 1} \tag{4.28}$$

$$\mathbf{S}_{i}^{-1} = \mathbf{T}^{-1} - \frac{\mathbf{T}^{-1}\mathbf{x}_{old}\mathbf{x}_{old}^{T}\mathbf{T}^{-1}}{\mathbf{x}_{old}^{T}\mathbf{T}^{-1}\mathbf{x}_{old} - 1} \tag{4.29}$$

Assuming that **x** is of size $Nx1$ and elements, and S_i of NxN, the number of operations are

$$\begin{aligned} \text{operations } &= N^2 \cdot \text{ multiplications } + 4 \cdot \text{ matrix multiplications} + \\ &2 \cdot \text{ matrix vector multiplications} + \\ &2 \cdot \text{ dot products } + 2 \cdot N^2 \cdot \text{ divisions} + \\ &(2 \cdot N^2 + 1) \text{ subtractions } + 1 \cdot \text{ addition} . \end{aligned} \tag{4.30}$$

4.2.3.1.10 Eigenvalues and Eigenvectors (hw)

To find the eigenvectors and eigenvalues of a symmetric matrix Gonzalez et. al. suggests the use of the Jacobi algorithm [47].

Figure 4.15 shows an diagram of how to perform one iteration of the jacobi algorithm. If the size of the matrix is NxN we need at least $\frac{N}{2}(N-1)$ iterations, but increasing the number of iterations will make the eigenvector and eigenvalue estimates even better.

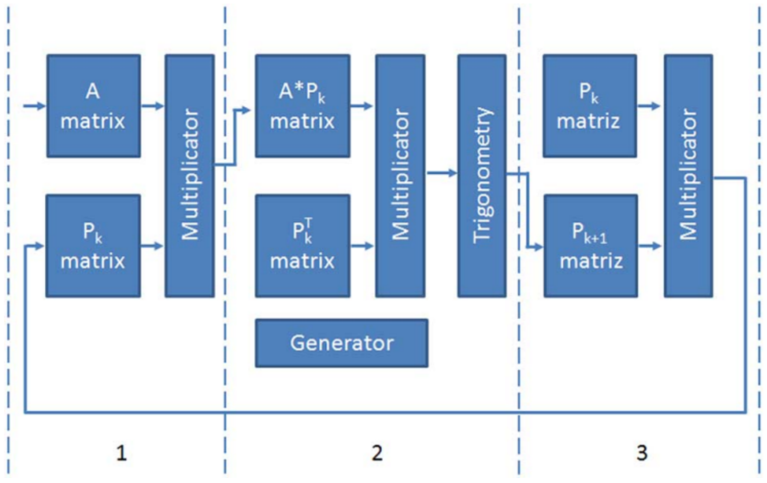


Figure 4.15: The three stages of the jacobi algorithm [47].

The the trigonometry block in figure 4.15 is $\frac{N}{2}$ processing cells consisting of the circuit shown in figure 4.16. The hardware implementation need 12 clock counts to propagate through the circuit, while the software implementation needs 13 operations.

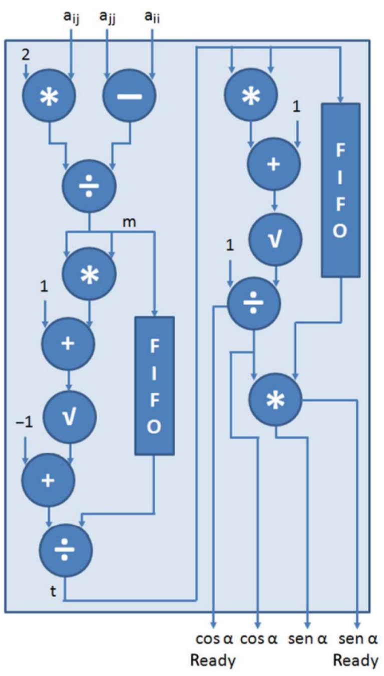


Figure 4.16: The three stages of the jacobi algorithm [47].

The estimated operation count of a hardware implementation of the jacobi algorithms is,

$$\begin{aligned}\text{operations } &= \frac{N}{2}(N-1)\cdot[3\cdot \text{ matrix multiplications } + 12]+\\ &\text{extra iterations } \cdot[3\cdot \text{ matrix multiplications } + 12].\end{aligned} \tag{4.31}$$

The estimated operation count of the software implementations is,

$$\begin{aligned}\text{operations } &= \frac{N}{2}(N-1)\cdot[3\cdot \text{ matrix multiplications } + \frac{N}{2}\cdot 13]+\\ &\text{extra iterations } \cdot[3\cdot \text{ matrix multiplications } + \frac{N}{2}\cdot 13].\end{aligned} \tag{4.32}$$

4.2.3.1.11 Eigenvalues and Eigenvectors (sw)

To find eigenvalues and eigenvectors for software algorithms, we use QR decomposition in an iterative manner.

First, we decompose the algorithms into a matrix consisting of orthogonal vectors and a compensating matrix. Then we change the order of the matrixes and produce a new matrix. The new matrix is again decomposed, and we follow the same procedure. This goes on for some iterations until we have a good enough estimate of the eigenvectors and eigenvalues.

4.2.3.1.12 Gaussian Elimination and the System of Equation Problem

The system of equation problem is to find the values of vector **x**, when the matrix **A** and vector **y** is known,

$$\mathbf{Ax} = \mathbf{y}. \tag{4.33}$$

x can be obtained by gaussian elimination of the matrix to the reduced row echelon form.

To obtain the reduced row echelon form we do row reductions. $R_{\text{new}} = R_{\text{old}} - c \cdot R_{\text{pivot row}}$. c is the constant found by dividing the element to be zeroed on the pivot element. The row subtraction is then $N+1$ number of subtractions, but since the elements in the columns with lower index than the pivot element are zeros, we only need a total of $\frac{N^2}{2} + \frac{N}{2} + N$ subtractions to achieve the row echelon form.

To achieve the reduced row echelon form and at the same time find the elements of the vector **x** we do the same procedure as described above, but starting at the element in position a_{NN}.

The operations needed to solve a system of equations by using the gaussian elimination method is therefore,

$$\text{operations } = 2 \cdot (N \text{ divisions } + \left(\frac{N^2}{2} + \frac{N}{2} + N \right) \cdot \text{ subtractions }) \tag{4.34}$$

Gaussian elimination of a matrix to obtain the row echelon form needs less operations,

$$\text{operations } = (N \text{ divisions } + \left(\frac{N^2}{2} + \frac{N}{2} \right) \cdot \text{ subtractions }) \tag{4.35}$$

4.2.3.1.13 Matrix Inversion

One way of doing matrix inversion is by using the LUP decomposition of a matrix, which is a decomposition of the matrix into three matrixes with the same dimensions as the original matrix,

$$\mathbf{A} = \mathbf{PLU} \tag{4.36}$$

P is a modifyed identity matrix that holds the row shifts made during the decompositions. **L** is a matrix containing elements only in the lower triangonal part with ones along the diagonal, and **U** is a matrix containing elements only in the upper triangonal part including the diagonal.

The **U** matrix is the row echelon form of the matrix, the lower matrix consist of the factors used in the gaussian elimination row reduction.

When having the LUP decomposite of the matrix A, we can find the matrix inversion by solving (**??**) for **z** and $\mathbf{a_i^{-1}}$, where $\mathbf{a_i^{-1}}$ is a column in the inverse matrix A^{-1}.

$$\begin{aligned} [\mathbf{L}][\mathbf{z}] &= [\mathbf{P}][\mathbf{e_i}] \\ [\mathbf{U}][\mathbf{a_i^{-1}}] &= [\mathbf{z}] \end{aligned} \tag{4.37}$$

The needed operations are,

$$\begin{aligned} \text{operations } = & \text{ gaussian elimination to echelon form } + \\ & \left(\frac{N^2}{2} + \frac{N}{2} \right) \cdot \text{ element copying } + \\ & 2 \cdot N \cdot \text{ system of equations problems} \end{aligned} \tag{4.38}$$

4.2.3.2 Binning

4.2.3.2.1 Spatial Binning

The three relevant equations needed for spatial binning are (4.39), (4.40) and (4.41).

$$\begin{gathered}a = \frac{1}{B}\sum_{i=0}^{B-1} x_i \\ \text{=> operations} = 1 \cdot \text{ divisions } + (B-1) \cdot \text{ additions}\end{gathered} \tag{4.39}$$

$$\begin{gathered}a = \frac{1}{\text{rest(framesamples}/B)} \sum_{i=0}^{\text{rest(framesamples}/B)-1} x_i \\ \text{=> operations} = 1 \cdot \text{ divisions } + (\text{rest(framesamples}/B) - 1) \cdot \text{ additions}\end{gathered} \tag{4.40}$$

$$\begin{gathered}a = \frac{1}{\text{rest(frames}/B)} \sum_{i=0}^{\text{rest(frames}/B)-1} x_i \\ \text{=> operations} = 1 \cdot \text{ divisions } + (\text{rest(frames}/B) - 1) \cdot \text{ additions}\end{gathered} \tag{4.41}$$

For doing spatial binning of spatial pixels inside a frame we follow this procedure:

1. To bin B spatial pixels we compute (4.39) band number of times.
2. We repeat step 1 for each bin in a frame.
3. If last bin contains less then B spatial pixels, we use (4.40) band number of times, to bin the last spatial bands.
4. We repeat step 1 to 3 for each frame.

The total number of operations needed to bin spatial pixels in the frames are estimated to be the following,

$$\begin{aligned}\text{operations } = \Bigg[& (4.39)_{\text{operations}} \cdot \text{ bands } \cdot \left\lfloor \frac{\text{framesamples}}{B} \right\rfloor + \\ & (4.40)_{\text{operations}} \cdot \text{ bands } \Bigg] \cdot \text{ frames .}\end{aligned} \tag{4.42}$$

Binning spatial pixel across frames, use the same procedure as described above, but exchange frames with framesamples, and vica versa. A consecuenze is that we use (4.41) instead of (4.40).

4.2.3.2.2 Spectral Binning

The purposed method of spectral binning uses SIMD operations that adds four successive elements using two operations, and uses weighted average thereby avoiding the division operations.

The equations needed for spectral binning are (4.43), (4.44) and (4.45).

$$\begin{gathered}a = \sum_{i=0}^{3} x_i \\ \text{=> operations } = 2 \cdot \text{SIMD operations}\end{gathered} \tag{4.43}$$

$$\begin{gathered}a = \sum_{i=0}^{\lfloor \frac{B}{4} \rfloor + \text{mod}(B,4)} x_i \\ \text{=> operations } = \left[\left\lfloor \frac{B}{4} \right\rfloor + \text{mod}(B,4) \right] \cdot \text{ additions}\end{gathered} \tag{4.44}$$

$$\begin{aligned} a = &\sum_{i=0}^{\text{rest(bands}/B)-1} x_i \\ => \text{operations} = &(\text{rest(framesamples}/B) - 1) \cdot \text{ additions} \end{aligned} \tag{4.45}$$

The procedure for doing the pruposed spectral binning is [51]:

1. Using (4.44) to add all elements to a bin, band number of times. In (4.43) The first $floor(B/4)$ set of elements are computed by (4.43).
2. Repeat step 1 for every bin in the frame that consist of B elements.
3. If the last bin contains less then B elements we bin the remaining elements by (4.45).
4. Repeat step 1 to 3 for every frame.

We estimate the number of operations to do spectral binning of a hyperspectral cube to be the following,

$$\begin{aligned} \text{operations } =&[(4.43)_{\text{operations}} \cdot \left\lfloor \frac{\text{bands}}{B} \right\rfloor \cdot \left\lfloor \frac{B}{4} \right\rfloor + \\ &(4.44)_{\text{operations}} \cdot \left\lfloor \frac{\text{bands}}{B} \right\rfloor \cdot + \\ &(4.45)_{\text{operations}}] \cdot \text{ frames } \cdot \text{ framesamples} . \end{aligned} \tag{4.46}$$

Usikker på om det er riktig med bands numer av tid, eller om det er bind bands, altså ciel(ands/B) - må se over denne!

4.2.3.3 Bad Pixel Detection and Correction

4.2.3.3.1 Bad Pixel Detection

Statistical Threshold Check (sw)

This detection strategy uses the equations (4.47), (4.48) and (4.49).

$$\begin{aligned} s^2 = &\frac{1}{N-1} \sum_{i=0}^{N-1} (x_i - \hat{x})^2 \\ => \text{operations } = &1 \cdot \text{ division } + N \cdot \text{ subtractions } + N \cdot \text{ multiplication} \end{aligned} \tag{4.47}$$

$$\begin{aligned} \hat{x} = &\frac{1}{N} \sum_{i=0}^{N-1} x_i \\ => \text{operations } = &1 \cdot \text{ division } + N \cdot \text{ additions} \end{aligned} \tag{4.48}$$

$$\begin{aligned} \hat{x}_{i+1} = &\hat{x}_i - \frac{x_{\text{oldest sample}}}{N} + \frac{x_{\text{new sample}}}{N} \\ => \text{operations } = &2 \cdot \text{ divisions } + 1 \cdot \text{ addition } + 1 \cdot \text{ subtraction} \end{aligned} \tag{4.49}$$

To detect bad pixels by the statistical threshold check method we do the following [22].

1. Compute the mean and covariance of the N neighbor pixels using (4.48) and (4.47).
2. Check if the pixel value is outside the acceptable range mean ± c) · variance, where c is chosen beforehand.
3. Repeat step 1 and 2 for each band in the spatial pixel

4. Repeat step 1 to 3 for each spatial pixel, but use (4.49) instead of (4.48)

The total number of operations are the following.

$$\begin{aligned} \text{operations } =&[(4.47)_{\text{operations}} + \\ &(4.49)_{\text{operations}} + \\ &1 \text{ addition } + 1 \text{ subtraction } + 1 \text{ multiplication } + \\ &1 \text{ comparison }] \cdot \text{bands} \cdot \text{ spatial pixels} + \\ &(4.48)_{\text{operations}} \cdot \text{bands} \end{aligned} \tag{4.50}$$

Correlation check (sw)

The procedure for neighbor correlation check can be summarized as the following [22].

1. In the neighbour correlation check we compute the correlation matrix of the spatial pixel in question and it's N neighbors. We do the correlation check for all the spatial pixel's spectral values
2. For every spectral value we check if there are X amount of bad correlations.
3. Update the correlation matrix to investigating the next pixel.
4. Repeat step 2 and 3 for all the following spatial pixels.

The total number of operations are,

$$\begin{aligned} \text{operations } =&[(\text{ compute correlation matrix of N elements } + N \text{ checks }) \cdot \text{ bands} + \\ &(\text{ update the correlation matrix } + \text{ checks }) \cdot \text{ bands } \cdot \text{ spatial pixels }. \end{aligned} \tag{4.51}$$

4.2.3.3.2 Bad Pixel Correction

Nearest Neighbour Replacement

This method is replacing the bad spectral value of a bad pixel with a neighbor's spectral value.
The method need one operation for copying an element.

Mean Replacement

This method replace the bad spectral pixel with the mean of it's 8 neighbor pixels.
(4.52) shows the procedure for computing the new spectral value, and the number of needed operations.

$$\begin{gathered} \hat{x} = \frac{1}{8}\sum_{i=1}^{8} x_i \\ => \text{operations } = 1 \cdot \text{ division } + 8 \cdot \text{ additions} \end{gathered} \tag{4.52}$$

Median Replacement

The procedure of the median replacement method is to replace a bad pixel with the median of it's neighbor pixels. We use the sorting algorithm ... to sort the neighbor pixels, and chose the middle value to replace the bad pixel.

Total number of operations are

$$\text{operations } = (\textbf{??})_{\text{operations}} + \text{ collect middle element.} \tag{4.53}$$

4.2.3.4 Smile and Keystone

Equations needed for smile and keytone correction are (4.54), (4.55), (4.56) and (4.57).

$$\begin{aligned} x &= a_{00} + a_{10}x_{ref} + a_{01}y_{ref} + a_{11}x_{ref}y_{ref} + a_{20}x_{ref}^2 + a_{02}y_{ref}^2 \\ y &= b_{00} + b_{10}x_{ref} + b_{01}y_{ref} + b_{11}x_{ref}y_{ref} + b_{20}x_{ref}^2 + b_{02}y_{ref}^2, \\ => \text{operations } &= 10 \cdot \text{ additions } + 16 \cdot \text{ multiplications} \end{aligned} \tag{4.54}$$

$$\begin{aligned} w1 &= (1 - d_x) \cdot (1 - d_y) \\ w2 &= (1 - d_x) \cdot d_y \\ w3 &= (1 - d_y) \cdot d_x \\ w4 &= d_y \cdot d_x \\ => \text{operations } &= 4 \cdot \text{ subtractions } + 4 \cdot \text{ multiplications} \end{aligned} \tag{4.55}$$

$$\begin{aligned} d_x &= x - x_{ref} \\ d_y &= y - y_{ref} \\ => \text{operations } &= 2 \cdot \text{ subtractions} \end{aligned} \tag{4.56}$$

$$\begin{aligned} v &= p1 \cdot w1 + p2 \cdot w2 + p3 \cdot w3 + p4 \cdot w4 \\ => \text{operations } &= 3 \cdot \text{ additions } + 4 \cdot \text{ multiplications} \end{aligned} \tag{4.57}$$

The procedure for doing smile and keystone correction can be summerized like follows [24]:

1. Compute the correct location of the spatial pixel with (4.54), which uses the known smile and keystone correction coefficients. In (4.54) (x, y) is the corrected pixel location, while (x_{ref}, y_{ref}) is the measured sample.
2. Use bipolar interpolation of four pixels neighbouring pixels, which are measures, to estimate a spatial pixel value for the new corrected pixel (x, y). We use (4.56) to find the distance between the corrected location and the sample location, (4.55) to find the weights dedicated to each neighbour pixel, and (4.57) to compute the new spatial value.
3. Repeat step 2 for each spectral value in the spatial pixels.
4. Repeat step 1 and 2 for each spatial pixel.
   of the four pixels that are nearest the corrected location.

The estimated number of operations needed in smile and keystone correction is,

$$\begin{aligned} \text{operations } =[&(4.54)_{\text{operations}} + \\ &(4.55)_{\text{operations}} + \\ &(4.56)_{\text{operations}} + \\ &(4.57)_{\text{operations}} \cdot \text{bands}] \cdot \text{ spatial pixels} \end{aligned} \tag{4.58}$$

4.2.3.5 Dimensional Reduction

4.2.3.5.1 PCA (sw) (hw)

The procedure to compute PCA is the following.

1. Compute the correlation matrix.
2. Compute the correlation matrix's eigenvalues and eigen vectors

3. Sort the eigenvalues.
4. Form a transformation matrix consisting of the N eigenvectors with highest eigenvalue.
5. Transform each spatial sample with the generated transformation matrix.

The number of operations needed for both the sw and hw version of PCA can be summerized in (4.59). The difference between the sw and hw version of PCA is how many operations needed to compute the elements of (4.59).

$$\begin{aligned}\text{operations } &= \text{correlation matrix computation } + \text{ eigenvector and eigenvalue computation } + \\ &\quad \text{sort eigenvalues } + \text{ matrix matrix multiplication}\end{aligned} \tag{4.59}$$

4.2.3.5.2 MNF (sw)

The implementation assumptions used in this thesis are based on [27]. The total number of operations is therefore obtained by interpreting the article.

4.2.3.5.3 ICA (sw)

The equation needed in ICA are (4.60), (4.61), and (4.62).

$$\begin{aligned}w_p &= \frac{1}{n}\sum_i^n x_i g(w^T x_i) - \frac{1}{n}\sum_i^n g'(w^T x_i)w \\ g(u) &= tanh(u), \\ g'(u) &= 1 - tanh^2(u) \\ \text{operations } &= \text{ spatial samples [dot products of band elements } + \text{ trigonometries } + \\ &\quad \text{subtraction } + (2 \cdot \text{ bands } + 1) \cdot \text{ multiplications]} + \\ &\quad 2 \cdot \text{ bands } \cdot \text{ division } + \text{ bands } \cdot \text{ subtraction}\end{aligned} \tag{4.60}$$

$$\begin{aligned}w_p &= w_p - \sum_{j=1}^{p-1}(w_p^T w_j)w_j \\ \text{operations } &= 2 \cdot \text{ dot products of band elements } + \text{ bands } \cdot \text{ subtractions}\end{aligned} \tag{4.61}$$

$$\begin{aligned}w_p &= \frac{w_p}{||w_p||} = \frac{w_p}{\sqrt{w_{p_1}^2 + w_{p_2}^2 + ... + w_{p_{\text{bands}}}^2}} \\ \text{operations } &= \text{ band } \cdot \text{ multiplications } + \text{ band } \cdot \text{ additions } + 1 \cdot \text{ sqrt } + \text{ band } \cdot \text{ divisions}\end{aligned} \tag{4.62}$$

The ICA algorithm can be implemented according to the following steps [28],

1. Compute the correlation matrix to x.
2. Whiten x with the inverse of the correlation matrix.
3. Initialize the de-mixing matrix W with random values.
4. Correct W by correcting one vectorelement w_p at a time using (4.60) and (4.61).
5. Normalize the corrected value w_p by using (4.62).

6. Check if $w_p^T w_{p+1} \approx 1$ or we have reached maximum number of iterations, if not return to 4) and continue correcting the same component, otherwise return to 4) but correct the next component,
7. compute the new representation of X, $S = WX$.

The total number of operations can be summarized as in equation (4.63), where P is the number of vectors in the transformation matrix W.

$$\begin{aligned} \text{operations} \; = \; & \text{correlation matrix computation} \; + \; \text{matrix inversion} \; + \\ & \text{matrix matrix multiplication} \; + P \cdot \; \text{iterations} \\ & [(4.60)_{\text{operations}} + \\ & (4.61)_{\text{operations}} + \\ & (4.62)_{\text{operations}} \;] + \; \text{matrix matrix multiplication} \end{aligned} \tag{4.63}$$

4.2.3.6 Georeferencing and Geometric Resampling

The proposed algorithm for georeferencing and geometric resampling is based on "Image registration and georeferencing with snapshot camera for the HYPSO mission", by Langer [29]. The purpose of the algorithm is to map the hyperspectral picture taken by the satellite to a location on earth and reconfigure the picture to have correct proportions compared to the Earth. The algorithm can be summarized into six steps.

1. Use (4.64) to find position and quaternion of the satellite when at the time of capturing a frame by linear interpolation. t_i denotes timestamps of the captured frames, and x_i and x_{i-1} are the closest positions or quaternions to the frames.
2. Find which direction each pixel is pointing by using (4.65) (4.66), (4.67), (4.68), and (4.69).
3. Find the intersection between the pixels view direction and the earth model to obtain Earth Centered Earth Fixed (ECEF) positions.
4. Find the corresponding coordinates in terms of longitude and latitude to the ECEF positions.
5. Determine the inverse mapping function
6. Do resampling of the image to correct the image to fit onto a map of the earth.

Step 1:

$$\begin{gathered} x_f = \frac{t^f - t^p_{i-1}}{t^p_i - t^p_{i-1}} x_{i-1} + \frac{t^p_i - t^f}{t^p_i - t^p_{i-1}} x_i \\ => \; \text{operations} \; = \; \text{frames} \; \cdot [4 \text{ subtractions } + 2 \text{ multiplications } + 1 \text{ additions }] \end{gathered} \tag{4.64}$$

Step 2:

$$\begin{aligned} R^f_b(q_{f_m}) = & \begin{bmatrix} 1 - 2\epsilon^2_{2m} - 2\epsilon^2_{3m} & 2(\epsilon_{1m}\epsilon_{2m} - \epsilon_{3m}\eta_m) & 2(\epsilon_{1m}\epsilon_{3m} + \epsilon_{2m}\eta_m) \\ 2(\epsilon_{1m}\epsilon_{2m} + \epsilon_{3m}\eta_m) & 1 - 2\epsilon^2_{1m} - 2\epsilon^2_{3m} & 2(\epsilon_{2m}\epsilon_{3m} - \epsilon_{1m}\eta_m) \\ 2(\epsilon_{1m}\epsilon_{3m} - \epsilon_{2m}\eta_m) & 2(\epsilon_{2m}\epsilon_{3m} + \epsilon_{1m}\eta_m) & 1 - 2\epsilon^2_{1m} - 2\epsilon^2_{2m} \end{bmatrix} \\ => \; \text{operations} \; = & \; 30 \cdot \text{multiplications} + 9 \cdot \text{subtractions} + 3 \cdot \; \text{additions} \end{aligned} \tag{4.65}$$

$$\begin{aligned} \hat{n}_m &= \mathrm{R}^f_b(q_{f_m}) R_g \hat{x} \\ c_m &= \mathrm{R}^f_b(q_{f_m}) \mathrm{R}_g \hat{z} \\ => \; \text{operations} \; &= 4 \cdot \; \text{vectormatrix multiplication} \; + (4.65)_{operations} \end{aligned} \tag{4.66}$$

$$\begin{aligned}\theta_n =& \tan^{-1}\left(2\left(\frac{n}{N}-\frac{1}{2}\right)\tan\frac{\alpha}{2}\right)\\ => \text{ operations } =& 2\cdot \text{ trigonometries } + 3\cdot \text{ divisions}\\ &+1\cdot \text{ subtractions } + 2\cdot \text{ multiplications}\end{aligned} \tag{4.67}$$

$$\begin{aligned}R(\hat{n},\theta) =& I + S(\hat{n})\sin\theta + S(\hat{n})^2(1-\cos\theta)\\ => \text{ operations } =& 2\cdot \text{ matrix addition } + 2\cdot \text{ matrix scalar multiplication } +\\ &2\cdot \text{ trigonometries } + 1\cdot \text{ matrix matrix multiplication } + 1\cdot \text{ subtraction}\end{aligned} \tag{4.68}$$

$$\begin{aligned}v_{m,n} =& \mathrm{R}(\hat{n}_m,\theta_n)c_m\\ => \text{ operations } =& \text{frames}\cdot[(4.66)_{operations} + \text{framesamples}\cdot[(4.67)_{operations}+\\ &(4.68)_{operations} + 1\cdot \text{ vectormatrix multiplication }]]\end{aligned} \tag{4.69}$$

STEP 3:

$$\begin{aligned}a &= v_x^2 + v_y^2 + \frac{v_z^2}{1-e^2}\\ b &= 2\left(p_x v_x + p_y v_y + \frac{p_z v_z}{1-e^2}\right)\\ c &= p_x^2 + p_y^2 + \frac{p_z^2}{1-e^2} - a_e^2\end{aligned} \tag{4.70}$$

$$=> \text{ operations } = 14\cdot\text{multiplications} + 3\cdot\text{divisions} + 6\cdot\text{additions} + 4\cdot\text{subtractions}$$

$$\begin{aligned}t_{min} &= -\frac{b}{2a} - \frac{1}{a}\sqrt{\frac{b^2}{4} - ac}\\ => \text{ operations } &= 5\cdot\text{multiplications} + 3\cdot\text{divisions} + 4\cdot\text{subtractions} + 1\cdot\text{sqrt}\end{aligned} \tag{4.71}$$

$$\begin{aligned}p_{m,n} &= p_{f_m} + t_{min}v_n\\ => \text{ operations } &= \text{frames}\cdot\text{framesamples}\cdot[(4.70)_{operations} + (4.71)_{operations}\\ &+ 3\cdot\text{multiplication} + 3\cdot\text{addition}]\end{aligned} \tag{4.72}$$

STEP 4:

$$\begin{aligned}\phi &= \tan^{-1}\left(\frac{z}{(1-e^2)\sqrt{x^2+y^2}}\right)\\ \lambda &= \tan^{-1}\left(\frac{y}{x}\right)\\ => \text{ operations } &= \text{frames}\cdot\text{framesamples}\cdot[2\cdot\text{trigonometries} + 4\cdot\text{multiplications}\\ &+ 2\cdot\text{divisions} + 1\cdot\text{additions} + 1\cdot\text{subtractions}]\end{aligned} \tag{4.73}$$

STEP 5:

This step consists of an optimization problem. In Equation 4.75, f_m^{-1} and f_n^{-1} are approximated as nth degree polynomials with coefficients $a_{i,j}$ and $b_{i,j}$ respectively, that are to be estimated by optimization.

$$\min_{\{a_{i,j}\}} \sum_{m=1}^{M} \sum_{n=1}^{N} |f_m^{-1}(x_{m,n}, y_{m,n}, \{a_{i,j}\}) - m|^2 \tag{4.74}$$

$$\min_{\{b_{i,j}\}} \sum_{m=1}^{M} \sum_{n=1}^{N} |f_n^{-1}(x_{m,n}, y_{m,n}, \{b_{i,j}\}) - n|^2 \tag{4.75}$$

STEP 6:

Apply f_m^{-1} and f_n^{-1} to every resample grid position, and perform nearest neighbor, or bilinear interpolation.

4.2.3.7 Target Detection

4.2.3.7.1 SAM (sw)

Equation (4.76) computes the SAM target detection value.

$$\begin{aligned} SAM &= \frac{(s^T x)^2}{(s^T s)(x^T x)} \\ \Rightarrow \text{operations} &= \; 3 \cdot \text{dot products of band elements+} \\ & \quad\; 2 \cdot \text{ multiplications } + 1 \cdot \text{ division} \end{aligned} \tag{4.76}$$

The procedure for computing SAM target detection image is as follows.

1. Calculate $(s^T s)$.
2. Compute (4.76) for every spatial pixel, but reuse $(s^T s)$ from step 1.

The total number of operations is,

$$\begin{aligned} \text{operations} &= \text{ dot product of band elements} \\ & \quad + [(4.76)_{operations} - \text{ dot product of band elements }] \cdot \text{ spatial pixels} \end{aligned} \tag{4.77}$$

4.2.3.7.2 SAM (hw)

Figure 4.17 is a possible hardware architecture for SAM pruposed by the author.

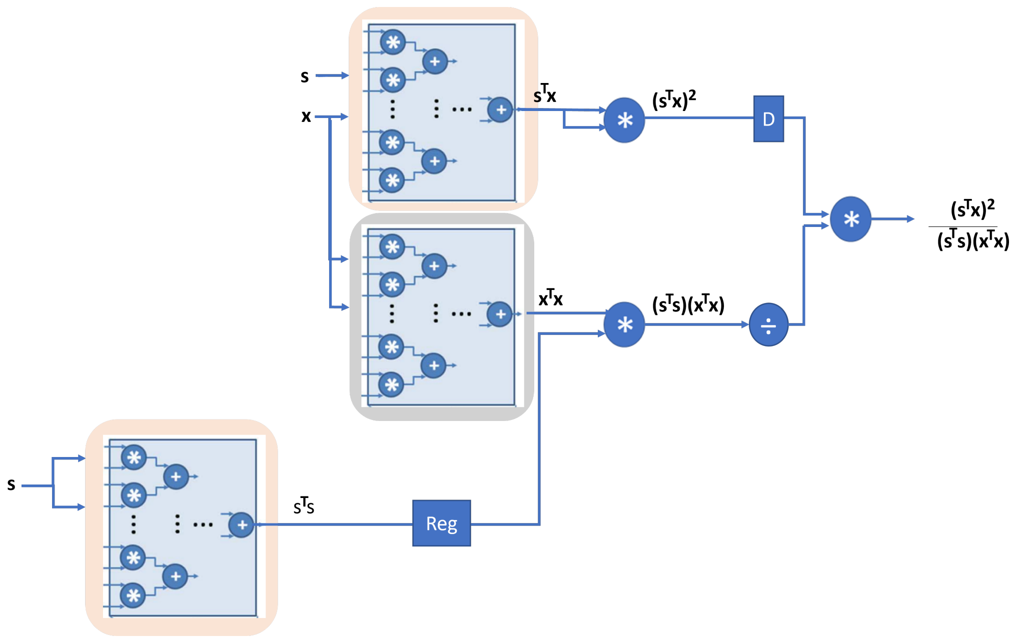


Figure 4.17: A possible hardware architecture for SAM.

The number of clock counts needed when using this hardware circuit is,

$$\text{operations} = \text{spatial samples} + (\text{ dot product } + 2\cdot \text{ multiplications } + 1\cdot \text{ division }) \tag{4.78}$$

4.2.3.7.3 CEM (sw)

Equation (4.83) computes the CEM target detection value.

$$\begin{aligned} CEM &= \frac{s^T R^{-1} x}{s^T R^{-1} s} \\ => \text{operations} &= 1 \text{ correlation matrix computation} + \\ &\quad 1 \text{ matrix inversion} + \\ &\quad 2\cdot \text{ matrix vector multiplication} + \\ &\quad 2\cdot \text{ dot products of band elements} + \\ &\quad 1\cdot \text{ division} \end{aligned} \tag{4.79}$$

The procedure for computing CEM target detection image is as follows.

1. Compute the correlation matrix and it's inverse
2. Calculate $(s^T R^{-1})$ and $(s^T R^{-1} s)$.
3. Compute (4.83) for every spatial pixel, but reuse the inverse correlation matrix, $(s^T R^{-1})$ and $(s^T R^{-1} s)$ from step 1 and 2.
4.

The total number of operations is,

$$\begin{aligned} \text{operations } &= 1 \text{ correlation matrix computation } + 1 \text{ matrix inversion } + \\ &\text{vector matrix multiplication } + \text{ dot product of band elements}+ \\ &\text{spatial pixels } \cdot (\text{ dot products of band elements } + \text{division}) \end{aligned} \tag{4.80}$$

4.2.3.7.4 ACE-R (sw)

Equation (4.81) computes the ACE-R target detection value for a pixel x.

$$ACE - R = \frac{(s^T R^{-1} x)^2}{(s^T R^{-1} s)(x^T R^{-1} x)} \tag{4.81}$$

The procedure for computing ACE-R target detection score for a whole hyperspectral image is as follows.

1. Compute the correlation matrix and it's inverse
2. Calculate $(s^T R^{-1})$ and $(s^T R^{-1} s)$.
3. For every spatial pixel, compute the matrix vector multiplication $x^T \cdot R^{-1}$, the dotproducts $(s^T R^{-1}) \cdot x$, and $(x^T R^{-1}) \cdot x$, and do two multiplications and one division.

The total number of operations is,

$$\begin{aligned} \text{operations } &= 1 \text{ correlation matrix computation } + 1 \text{ matrix inversion } + \\ &\text{vector matrix multiplication } + \text{ dot product of band elements}+ \\ &\text{spatial pixels } \cdot [1 \cdot \text{ matrix vector multiplication } + \\ &2 \cdot \text{ dot products of band elements } + 2 \cdot \text{ multiplication } + 1 \cdot \text{ division }] \end{aligned} \tag{4.82}$$

4.2.3.7.5 SAM, CEM, ACE-R, and ASMF (hw)

The algorithms SAM, CEM and ACE-R are shown in equations (4.76), (4.83) and (**??**), while ASMF is given as follows,

$$ASMF = \frac{s^T R^- 1x}{s^T R^- 1s} \cdot \left| \frac{s^T R^- 1x}{x^T R^- 1s} \right|^n . \tag{4.83}$$

The hardware prupoosed circuit from "A reconfigurable multi-mode implementation of hyperspectral target detection algorithms", uses the following number of operations to complete [18].

$$\text{operations} = \text{ spatial pixels } \cdot [3 \cdot \text{ bands } + \text{ division } + 3] + 2 \cdot \text{ bands } + \text{ division } + 3, \tag{4.84}$$

4.2.3.8 Anomaly Detection

4.2.3.9 GRX-R (sw)

The equation that computes GRX-R scores is given in (4.85).

$$\begin{aligned} GRX - R &= (\mathbf{x} - \boldsymbol{\mu})^T \hat{R}^{-1} (\mathbf{x} - \boldsymbol{\mu}) \\ \text{operations } &= \text{ correlation matrix computation } + \text{ matrix inversion } + \\ &\text{mean of spatial pixels } + \text{ bands } \cdot \text{ subtraction } + \\ &\text{vector matrix multiplication } + \text{ dot product of band elements} \end{aligned} \tag{4.85}$$

The procedure for computing the GRX-R score to the hyperspectral cube the following [1].

1. Compute the GRX-R score of the first spatial pixel using (4.85).
2. When computing the GRX-R score the next spatial pixels we reuse the calculations from step 1, except exchanging **x**.

The total number of operations are

$$\begin{aligned}\text{operations } &= (4.85)_{\text{operations}} + \text{ spatial pixels [bands } \cdot \text{ subtractions } + \\ &1 \cdot \text{ vector matrix multiplication } + 1 \cdot \text{ dot product of band elements]}\end{aligned} \tag{4.86}$$

4.2.3.10 LRX (sw)

The LRX-R score is given in (4.87).

$$\begin{aligned} LRX - R &= (\mathbf{x} - \boldsymbol{\mu}_{\mathbf{local}})^T \hat{R^{-1}}_{local} (\mathbf{x} - \boldsymbol{\mu}_{\mathbf{local}}) \\ \text{operations } &= \text{ correlation matrix computation } + \text{ matrix inversion } + \\ &\text{mean of local spatial pixels } + \\ &\text{bands } \cdot \text{ subtraction } + \text{ vector matrix multiplication } + \\ &\text{dot product of band elements}\end{aligned} \tag{4.87}$$

The procedure for computing the LRX-R score to the hyperspectral cube the following [1].

1. Compute the LRX-R score of the first spatial pixel using (4.85).
2. Update the local mean and local correlation matrix.
3. Calculate the LRX-R score for the new pixel.

The total number of operations are

$$\begin{aligned}\text{operations } &= (4.87)_{\text{operations}} + \text{ spatial pixels [update local mean } + \\ &\text{update inverse of correlation matrix } + \text{ bands } \cdot \text{ subtractions } + \\ &1 \cdot \text{ vector matrix multiplication } + 1 \cdot \text{ dot product of band elements]}\end{aligned} \tag{4.88}$$

4.2.3.11 FrFT-RX (sw)

Equations needed for FrFT-RX anomaly detection are (4.89), (4.90), (4.91).

$$\begin{aligned}\mathbf{x}_p(u) &= (1/d)\sum_{s=1}^{d}\mathbf{x}(s) + K_p(s,u) \\ \text{operations } &= 1 \cdot \text{ division } + \text{ bands } \cdot \text{ additions}\end{aligned} \tag{4.89}$$

$$\begin{aligned}K_p(s,u) &= A\phi \ \exp[j\pi(s^2 \cot \phi - 2su \csc \phi + u^2 \cot \phi)] \\ \phi &= p\pi/2 \\ \text{operations } &= 10 \cdot \text{ multiplications } + 3 \cdot \text{ trignometries } + 1 \cdot \text{ division } + \\ &1 \cdot \text{ subtraction } + 1 \cdot \text{ addition } + 1 \cdot \text{ exp}\end{aligned} \tag{4.90}$$

$$\begin{aligned} A_\phi &= \frac{\exp[-j\pi \text{ sgn}(\sin \phi)/4 + j\phi/2]}{|\sin \phi|^{1/2}} \\ \phi &= p\pi/2 \\ \text{operations } &= 3 \cdot \text{ division } + \text{ exp } + 3 \cdot \text{ multiplication} + \\ &\quad 1 \cdot \text{ trignometries } + 1 \cdot \text{ absolute value } + 1 \cdot \text{ sqrt } + \text{ sign} \end{aligned} \tag{4.91}$$

$$\begin{aligned} E &= -\sum_{i=0}^{M-1} q_i \log(q_i) \\ \text{operations } &= M \cdot (\text{ additions } + 1 \cdot \text{logarithmic}+ \\ &\quad 1 \cdot \text{ multiplication }) + 1 \cdot \text{ multiplication} \end{aligned} \tag{4.92}$$

$$\begin{aligned} \text{FrFE}_p &= E \cdot \mathbf{X}_p \\ \text{operations } &= \text{ spatial pixels } \cdot \text{ bands } \cdot \text{ multiplications} \end{aligned} \tag{4.93}$$

$$\begin{aligned} lp &= \text{ arg } \max_p \text{FrFE}_p \\ \text{operations } &= M \cdot \text{ checks} \end{aligned} \tag{4.94}$$

The procedure stated in the following list [31].

1. We use (4.89), (4.90) and (4.91) to obtain a spectral value of one spatial pixel in the fractional domain.
2. Repeat step 1 for every spectral band in a spatial pixel.
3. Repeate step 1 and 2 M times to obtain the spatial pixel in M different fractional domains.
4. Compure E according to (4.92), where we assume q_i are known. In case q_i is not known we discard the computation of q_i, as we assume it is not a dominant term in this computation.
5. Compute FrFE_p with (4.93).
6. Compute (4.94).
7. Do RX-R anomaly detection on the cube represented in the optimal p fractional domain.

$$\begin{aligned} \text{operations } = &\text{ spatial pixels } \cdot M \cdot \text{ bands} \cdot \\ &[(4.89)_{\text{operations}} + (4.90)_{\text{operations}} + (4.91)_{\text{operations}}] + \\ &(4.92)_{operations} + (4.93)_{operations} + \\ &(4.94)_{operations} + (4.86)_{operations} \end{aligned} \tag{4.95}$$

4.2.3.12 Collabrative representation for hyperspectral anomaly detection (CRD) (sw)

We use equation (4.96) and (4.97) to calculate CRD scores.

$$\begin{aligned} \hat{\alpha} &= (\mathbf{X_s^T X_s} + \lambda \mathbf{I})^{-1} \mathbf{X_s^T x} \\ \text{operations } &= \text{ matrix matrix multiplication } + \text{ band } \cdot \text{ additions} + \\ &\quad \text{matrix inversion } + 2 \cdot \text{ vector matrix multiplication} \end{aligned} \tag{4.96}$$

$$\begin{aligned} CRD &= ||\mathbf{x} - \mathbf{X}_s \hat{\alpha}||_2 \\ \text{operations } &= \text{ vector matrix multiplication} + \\ &\quad \text{bands } \cdot [\text{ subtraction } + \text{ distance computation }] \end{aligned} \tag{4.97}$$

The procedure for computing the CRD scores is the following [32].

1. Calculate α according to (4.96), where s denotes how many neighbour pixels the are in the omitting matrix X_S.
2. Compute the CRD score using (4.97).
3. Repeat step 1 and 2 for every spatial sample, but use inverse matrix updating instead of calculating $(\mathbf{X_s^T X_s} + \lambda \mathbf{I})^{-1}$ form scratch.

The total number of operations are given in equation (4.98)

$$\begin{aligned}\text{operations } = & (4.96)_{operations} + \\ & \text{spatial pixels } [\cdot (4.97)_{operations} + \\ & \text{update inverse matrix } + 2 \cdot \text{ vector matrix multiplication }]\end{aligned} \tag{4.98}$$

4.2.3.13 Fast morphological and fuided filters (F-MGD) (hw)

The prupused implementation of F-MGD is highly paralleled. The parallell computation is described in the following list [33].

1. Do morphological opening and closing reconstruction iterativly, for each spectral band in parallell.
2. Extract anomalies with a differential operation for each band.
3. Do a feature clustering procedure of each band.
4. Average the outputs of the bands feature clustering procedure to obtain the final score.

By interpreteing [33], we assume that we construct band number of pipeline operating on one pixel every clock cycle. As stage 1 is an iterative process we assume 20 stages to complete that task. We use something that resambles a sliding window (kernal structing element) of size rxr. In each iterative stage we assume $2 \cdot (r-1)$ operations. Stage 2 is subtraction is therefore considered to use 1 clock cycle. Stage 3 consist of 8 stages, each assumed to be one clock cycle long.

The total number of operations can be estimated to the following.

$$\text{operations } = 20 \cdot (r-1) + \text{ subtraction (hw) } + 8 + \text{ mean (hw) } + \text{ spatial pixels} \tag{4.99}$$

4.2.4 Classification

4.2.4.1 SVM (sw)

The equations needed for SVM classification are (4.100), (4.101), (4.102) and (4.103), where $t_{m\#svn}$ and $t_{n\#svn}$ are the subsets of training samples for class number m and n. γ and b is predefined or foreknown parameters, and T is the number of classes.

$$\begin{aligned} f(\mathbf{x}) = & \sum_{i \epsilon (t_{m\#svn} + t_{n\#svn})} \alpha_i y_i K(\mathbf{x_i}, \mathbf{x}) + b \\ \text{operations } = & \text{ number of support vectors } \cdot [2 \cdot \text{ multiplications } + \\ & \text{kernel functions } + \text{ additions }]\end{aligned} \tag{4.100}$$

$$\begin{aligned} K(\mathbf{x_i}, \mathbf{x}) = & \exp(-\gamma ||\mathbf{x_i} - \mathbf{x}||^2) \\ ||\mathbf{x_i} - \mathbf{x}||^2 = & (x_{i_1} - x_1)^2 + (x_{i_2} - x_2)^2 + ... + (x_{i_{\text{bands}}} - x_{\text{bands}})^2 \\ \text{operations } = & \text{ exponential operation } + \text{ multiplication } + \text{ distance } - \text{ sqrt}\end{aligned} \tag{4.101}$$

$$\begin{aligned} S_i(\mathbf{x}) &= \sum_{j=1}^{T} \text{sgn}\left\{f_{ij}(\mathbf{x})\right\}, j \neq i. \\ \text{operations } &= (T-1) \cdot \text{ additions} \end{aligned} \tag{4.102}$$

$$\begin{aligned} \omega^* &= \text{ arg max}\{S_i(\mathbf{x})\} \\ \text{operations } &= (T-1) \cdot \text{ comparisons} \end{aligned} \tag{4.103}$$

The procedure for doing SVM classification is the following [35].

1. Compute (4.100) using (4.101) which class a pixel belong to, compearing two classes.
2. Repeat step 1 untill we have compeard every class with eachother ($T^2/2 - T/2$ reperations).
3. Use (4.102) and (4.103) to determined the class of the pixel.
4. Repeat step 1 to 3 for every spatial pixel.

The number of operations needed to do the classification is

$$\begin{aligned} \text{operations } = \text{ spatial pixels } [(T^2/2 - T/2) \cdot [(4.100)_{\text{operations}} + \\ (4.101)_{\text{operations}}] + (4.102)_{\text{operations}} + (4.103)_{\text{operations}}] \end{aligned} \tag{4.104}$$

4.2.4.2 CCSDS123 Compression

4.2.4.2.1 CCSDS123-B1 (sw)

The article "An Efficient Real-Time FPGA Implementation of the CCSDS-123 Compression Standard for Hyperspectral Images" by Fjeldtvedt et. al. describe a possible implementation of the CCSDS123 lossless compression algorithm [52]. The algorithm start with a prediction of the samples, it computes the residual between the actual pixel and the predicted, and then encode the residual. Since this is an rough operation count estimate of the algorithm, we look at the algorithm as if there was no boundary pixels, or special cases.

In the following equations there are several pre-chosen constants, which includes D, P, Ω, v_{min}, v_{max}, t_{inc}, R, U_{max}, γ_0, γ^*, K, s_{min}??, s_{max}??, s_{mid}??, ω_{min}, ω_{max}, t_{inc}, N_x.

To compute the resudol to be encoded we use equation (4.105), (4.106), (4.107) and (4.108).

$$\begin{aligned} \delta &= |\Delta_z(t)| + \theta_z(t) \\ \theta_z(t) &= min\{\hat{s}_z(t) - s_{min}, s_{max} - \hat{s}_z(t)\} \\ => \text{ operations } &= 1 \cdot \text{ absolute value } + 1 \cdot \text{ addition } + \\ & 2 \cdot \text{ subtraction } + 1 \text{ check} \end{aligned} \tag{4.105}$$

$$\begin{aligned} \Delta_z(t) &= s_z(t) - \hat{s}_z(t) \\ => \text{ operations } &= \text{ subtraction} \end{aligned} \tag{4.106}$$

$$\begin{aligned} \tilde{s}_z(t) &= \left\lfloor \frac{\text{clip}(\tilde{s}_{round}(t) + 2s_{mid} + 1, (2s_{min}, 2s_{max} + 1))}{2} \right\rfloor \\ => \text{ operations } &= 3 \cdot \text{ addition } + 3 \cdot \text{ multiplication } + \\ & 1 \cdot \text{ clip } + 1 \cdot \text{ division } + 1 \cdot \text{ round down} \end{aligned} \tag{4.107}$$

$$\begin{aligned}\tilde{s}_{round}(t) &= \left\lfloor \frac{\text{mod}^*_R[\hat{d}_z(t) + 2^{\Omega}(\sigma_z(t) - 4s_{mid}]}{2^{\Omega+1}} \right\rfloor \\ \text{mod}^*_R[x] &= ((x + 2^{R-1})\text{mod}2^R) - 2^{R-1} \\ => \text{ operations } &= 3\cdot \text{ addition } + 4\cdot \text{ subtraction } + 2\cdot \text{ multiplications } + \\ & 1\cdot \text{ division } + 1\cdot \text{ round down } + 1\cdot \text{ modulo operation}\end{aligned} \tag{4.108}$$

$$\begin{aligned} d^k_{z,y,x} &= 4\cdot s_{z-k,y,x} - \sigma_{z-k,y,x} \\ => \text{ operations } &= 1\cdot \text{ multiplication } + 1\cdot \text{ subtraction}\end{aligned} \tag{4.109}$$

$$\begin{aligned}\sigma_{z,y,x} &= s_{z,x,y-1} + s_{z,x-1,y-1} + s_{z,x-1,y} + s_{z,x+1,y-1} \\ => \text{ operations } &= 3\cdot \text{ addition}\end{aligned} \tag{4.110}$$

$$\begin{aligned} W_z(t+1) &= \text{clip}(W_z(t) + \Delta W_z(t), \{\omega_{min}, \omega_{max}\} \\ => \text{ operations } &= 1\cdot \text{ clip } + 1\cdot \text{ addition}\end{aligned} \tag{4.111}$$

$$\begin{aligned}\Delta W_z(t) &= \left\lfloor \frac{1}{2}(\text{ sign}^+[e_z(t)]\cdot 2^{-\rho(t)}\cdot U_z(t) + 1) \right\rfloor \\ e_z(t) &= 2s_z(t) - \tilde{s}_z(t) \\ => \text{ operations } &= 3\cdot \text{ multiplications } + 1\cdot \text{ addition } + 1\cdot \text{ shift } + \\ & 1\cdot \text{ sign } + 1\cdot \text{ division } + 1\cdot \text{ round down } + 1\cdot \text{ subtraction}\end{aligned} \tag{4.112}$$

$$\begin{aligned}\rho(t) &= \text{clip}(v_{min} + \left\lfloor \frac{t - N_x}{t_{inc}} \right\rfloor, \{v_{min}, v_{max}\}) + D + \Omega \\ => \text{ operations } &= 1\cdot \text{ clip } + 3\cdot \text{ addition } + 1\cdot \text{ round down } + \\ & 1\cdot \text{ subtraction } + 1\cdot \text{ division}\end{aligned} \tag{4.113}$$

$$\begin{aligned} u_z(t) &= \left\lfloor \frac{\delta_z(t)}{2^{k_z(t)}} \right\rfloor, \\ r_z &= \delta_z(t) \text{ mod } 2^{k_z(t)} \\ => \text{ operations } &= 1\cdot \text{ division } + 1\cdot \text{ round down} \\ & 1\cdot \text{ shift } + 1\cdot \text{ modulus}\end{aligned} \tag{4.114}$$

$$\begin{aligned} 2^i &\le \frac{\Sigma_z(t) + \left\lfloor \frac{49}{2^7}\Gamma(t) \right\rfloor}{\Gamma(t)} \\ => \text{ operations } &= 2\cdot \text{ division } + 1\cdot \text{ addition } + \\ & 1\cdot \text{ multiplication } + 1\cdot \text{ round down}\end{aligned} \tag{4.115}$$

$$\begin{aligned}\Gamma(t) &= \left\lfloor \frac{\Gamma(t-1) + 1}{2} \right\rfloor \\ => \text{ operations } &= 1\cdot \text{ division } + 1\cdot \text{ addition } + \\ & 1\cdot \text{ round down}\end{aligned} \tag{4.116}$$

$$\begin{aligned}\Sigma(t) &= \left\lfloor \frac{\Sigma_z(t-1) + \delta_z(t-1) + 1}{2} \right\rfloor \\ => \text{ operations } &= 1\cdot \text{ division } + 2\cdot \text{ addition } + \\ & 1\cdot \text{ round down}\end{aligned} \tag{4.117}$$

The procedure of computing the CCSDS123 algorithm is described in the following list.

1. Compute the residual δ with (4.105), (4.106), (4.107) and (4.108).
2. Compute $\hat{d}$ by the dot product of a vector U consisting of $P+3$ number of d values and the weight vector W. The d vectors are computed by (4.109) and (4.110), while the weight vector by (4.111), (4.112) and (4.113).
3. Compute the quotient and residual pair $(u_z(t), r_z)$ with (4.114) where k is the largest i in the range of bit resolution satisfying the equation (4.115), using (4.116) and (4.117).

The total number of operations are,

$$\begin{aligned} \text{operations} \;=\; & \text{spatial pixels} \cdot \text{bands} \cdot [(4.105)_{\text{operations}} + \\ & (4.106)_{\text{operations}} + \\ & (4.107)_{\text{operations}} + \\ & (4.108)_{\text{operations}} + 1 \cdot \text{ dot product of P+3 elements} + \\ & (4.109)_{\text{operations}} \cdot (P+3) + \\ & (4.110)_{\text{operations}} + \\ & (4.111)_{\text{operations}} + \\ & (4.112)_{\text{operations}} + \\ & (4.113)_{\text{operations}} + \\ & (4.114)_{\text{operations}} + \\ & (4.115)_{\text{operations}} + \text{ bit resolution} \cdot \text{ checks} + \\ & (4.116)_{\text{operations}} + \\ & (4.117)_{\text{operations}} \,] \end{aligned} \tag{4.118}$$

4.2.4.2.2 CCSDS123-B1 (hw)

An hardware implementation of the CCSDS123 lossless compression algorithm are given in by [52]. The clock count to the algorithm processing two samples is shown in figure 4.18.

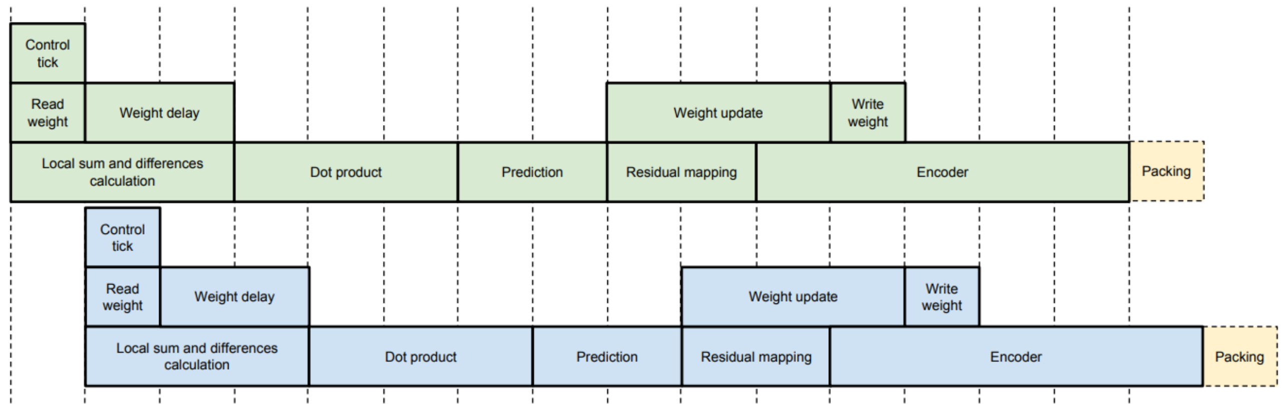


Figure 4.18: The timing diagram of the pruposed hardware implementation of CCSDS123 B1 [52].

In figure 4.18 the dot product stage is set to 4 clock cycles. However, this varies according to the variable

P. The clock cycles of the dot product stage is $\lceil log_2(P) + 1 \rceil$. The last stage is also depending on the encoded word, however it is assumed to process one sample at each clock cycle. The total number of clock cycles are therefore,

$$\text{Operations} = \text{ frames } \cdot \text{ framesamples } \cdot \text{ bands } + 12 + \log(P) + 1 \tag{4.119}$$

4.2.4.2.3 CCSDS123-B2 (sw)

This algorithm is similar to the one above, but provides even better compression rate, on the behalf of data loss.

The article "Hardware Implementation of the CCSDS 123.0-B-2 Near-Lossless Compression Standard Following an HLS Design Methodology" describes the CCSDS123 B2 algorithm, and compare the CCSDS123 B2 algorithm with the CCSDS123 B1 algorithm, decribed in the previous subsection [53]. The main architectural differences between the two versions of the CCSDS123 compression algorithm is that CCSDS123 B2 adds two additional computation blocks to the operation chain. Figure 4.19 shows the operation chaing of CCSDS123 B2, where the boxes marked in red are the additions compared to the CCSDS123 B1 algorithm.

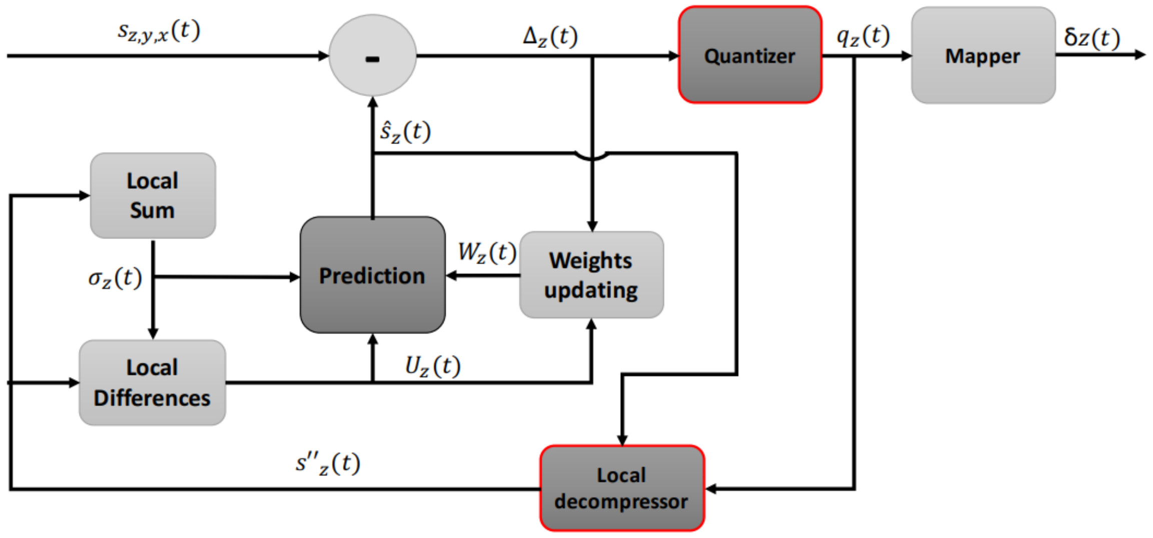


Figure 4.19: The processing chain of CCSDS123 B2 which is similar to the processing of CCSDS123 B1, but includes the additional processing stages marked in red [53].

Even though, the main differences is covered by including the extra processing stages, there are small changes throughout the circuit. These changes are covered in detail in a master thesis by Boothby, nevertheless, we will disregard them in this operation count estimation as they are assumed to be insignificant [37].

The extra quentizer stage computes a quentizer using (4.120), (4.121) and (4.122), while the extra local decompressor stage uses (4.123), (4.124) and (4.125). In the mentioned equations Psi_z, r_z, m_z are predefined variables.

$$\begin{aligned} q_z(t) = &\ \text{sign}(\Delta_z(t))\left\lfloor \frac{|\Delta_z(t)| + m_z(t)}{2m_z(t)+1} \right\rfloor \\ => \text{ operations } &= 1\cdot \text{ sign } + 2\cdot \text{ multiplications } + 1\cdot \text{ absolute value } + \\ & 2\cdot \text{ additions } + 1\cdot \text{ division } + 1\cdot \text{ round down} \end{aligned} \tag{4.120}$$

$$\begin{aligned} \delta &= |\Delta_z(t)| + \theta_z(t) \\ \theta_z(t) &= min\{\left\lfloor \frac{\hat{s} - s_{min} + m_z(t)}{2m_z(t)+1} \right\rfloor, \left\lfloor \frac{s_{max} - \hat{s} + m_z(t)}{2m_z(t)+1} \right\rfloor \\ => \text{ operations } &= 1\cdot \text{ absolute value } + 4\cdot \text{ addition } + 2\cdot \text{ subtractions } + \\ & 1\cdot \text{ multiplication } + 2\cdot \text{ round downs } + 1\cdot \text{ check} \end{aligned} \tag{4.121}$$

$$\begin{aligned} m_z(t) &= min(a_z, \left\lfloor \frac{r_z|\tilde{s_z}(t)}{2^D} \right\rfloor) \\ => \text{ operations } &= 1\cdot \text{ absolute value } + 1\cdot \text{ multiplication } + \\ & 1\cdot \text{ division } + 1\cdot \text{ round down } + 1\cdot \text{ check} \end{aligned} \tag{4.122}$$

$$\begin{aligned} s_z''(t) &= \left\lfloor \frac{\tilde{s}_z''(t) + 1}{2} \right\rfloor \\ => \text{ operations } &= 1\cdot \text{ addition } + 1\cdot \text{ division } + 1\cdot \text{ round down} \end{aligned} \tag{4.123}$$

$$\begin{aligned} \tilde{s}_z''(t) &= \left\lfloor \frac{4(2^{\Theta} - \phi_z \cdot (s_z'(t)\cdot 2^{\Omega} - sgn(q_z(t))\cdot m_z(t)\cdot \Psi_z \cdot 2^{\Omega-\Theta}) + \phi_z \cdot \check{s}_z(t) - \phi_z \cdot 2^{\Omega+1}}{2^{\Omega+\Theta+1}} \right\rfloor \\ => \text{ operations } &= 8\cdot \text{ multiplications } + 3\cdot \text{ subtractions } + 3\cdot \text{ additions } + \\ & 1\cdot \text{ sign } + 1\cdot \text{ division } + 1\cdot \text{ round down} \end{aligned} \tag{4.124}$$

$$\begin{aligned} s_z'(t) = &\ \text{clip } (\tilde{s}_z(t) + q_z(t)(2m_z(t)+1), \{s_{min}, s_{max}\}) \\ => \text{ operations } &= 2\cdot \text{ addition } + 2\cdot \text{ multiplication } + 1\cdot \text{ clip} \end{aligned} \tag{4.125}$$

The operation estimate of the CCSDS123 B2 algorithm is,

$$\begin{aligned} \text{operations } = &\ (4.118)_{\text{operations}} + \\ & \text{spatial samples } \cdot \text{ bands } [(4.120)_{\text{operations}} + \\ & (4.121)_{\text{operations}} + \\ & (4.122)_{\text{operations}} + \\ & (4.123)_{\text{operations}} + \\ & (4.124)_{\text{operations}} + \\ & (4.125)_{\text{operations}}] \end{aligned} \tag{4.126}$$

4.2.4.2.4 CCSDS123-B2 (hw)

The pruposed cuircuite described in "Hardware Implementation of the CCSDS 123.0-B-2 Near-Lossless Compression Standard Following an HLS Design Methodology", generate a prediction residual every 7 clock cycle, and have a latency of 13 clock cycles. The total number of operations is therefore the following [53].

$$operations = frames * framesamples * bands * 7 + 13 \tag{4.127}$$

Chapter 5

Verification

In this chapter we verify the algorithms' run time estimations.

5.1 Verification of Run Time Estimations

Estimating the run time of an algorithm theoretically means that we guess a run time based on how we perceive the algorithm. When algorithms are described with a high level of abstraction, the implementation details can be interpreted in numerous ways. If we choose to estimate the run time of an algorithm on implementations that are more inefficient compared to other possible implementations, the design space exploration will provide an unrealistic insight. One way to handle the mentioned problem is to do thoughtful research about what kinds of different implementations exist for the various algorithms. Nevertheless, this can potentially violate the concept of the design space exploration, namely that it should be rough and time friendly.

A faster and less exhausting alternative for ensuring the run time estimations are realistic is verifying the run time estimate with performance studies of the algorithm. If the estimated run time is slower than a measured run time, we know there is a faster and thereby better implementation than the one we currently are using. If the run time estimate is faster than the measured run time, we either conclude that our assumed implementation is better than the measured implementation, that the measured implementation suffers from overheads related to the operating platform or compiler optimization (in this case compiler anti optimization), or that our run time estimate is wrong because we have missed to considered timely parts of the algorithm. Even though there is a chance the run time estimates are incorrect, we still use them.

The estimated run times should be faster than the measured run time because we assume ideal conditions. In the run time estimate we assume that all samples are available at an instance, there are no memory overheads or instruction delays, all instructions that seem like basic arithmetic or logical instructions will only demand one operation count, and we assume that overflow is not a problem. The mentioned assumptions are unrealistic, but determining the impact these features have on computers, in general, is impossible. Each of the mentioned features depends on the computers' unique platform, compilers, and current operational states. Consequently, to make the run time estimation simple, we assume ideal conditions, and the run time estimates should therefore be faster than any measured run time.

Table 5.1 shows an overview of estimated run times and the measured run times of implemented algorithms when the data is of a given size and the operating platform uses a given frequency. The table also displays where the run time measurements are collected. Some run time measurements are found in performance studies, while some are generated by code already available for HYPSO.

Unfortunately, table 5.1 does not hold code run time measurement for all the algorithms. The algorithms that lack run times estimates are those that are:

- Suggested by the author, which includes the spatial binning algorithm and the hardware implementation of SAM.
- Hardware implementation of algorithms where the performance study states in detail the number of operations needed for each data point. This includes the hardware implementation of the target detection algorithms CEM, Adoptive Coherence/Cosine Estimator (ACE), and ASMF, and the compression algorithm CCSDS123-B1.
- Algorithms with performance studies that focus on the quality metric instead of the run time of the algorithm, and HYPSO does not possess an implementation of the algorithm. The algorithms in this category are bad pixel detection and correction and radiometric calibration.

Table 5.1: Estimated and recorded run times for some algorithmic implementations, given a data input size and operation frequency. The recorded run time samples are collected from the listed sources.

Algorithm	**Inputted data size**	**Operation frequency**	**Recorded run time**	**Estimated run time**	**Run time source**
Spectral binning	1x1000x1216	667 MHz	0.008 sec	0.001 sec	[51]
Smile and keystone correction	956x684x120	667 MHz	8.76 sec	0.81 sec	[54]
PCA (sw)	350x350x224	2.6 GHz	15.02 sec	1.81 sec	[40]
MNF (sw)	1x1600x160	4.7 GHz	23.15 sec	1.03 sec	[55]
ICA (sw)	956x684x1080	2.7 GHz	407.5?	59 sec	[56]
SAM (sw)	956x684x120	2.7 GHz	0.38 sec	0.12 sec	Internal code
CEM (sw)	956x684x120	2.7 GHz	84,87 sec	17.43 sec	Internal code
ACE (sw)	956x684x120	2.7 GHz	114,51 sec	24.40 sec	Internal code
GRX (sw)	64x64x169	2.8 GHz	1.80 sec	0.30 sec	[32]
LRX (sw)	64x64x169	2.8 GHz	142.43 sec	56.71 sec	[32]
CRD (sw)	64x64x169	2.8 GHz	62.84 sec	10.64 sec	[32]

Chapter 6

Results and Discussion

This chapter presents the result obtained in this design space exploration and discusses them. This section is divided into three parts. The first part shows and discusses the run time of the various algorithms in this design space exploration and will provide an intuition of what onboard data processing pipelines can be interesting to check out.

The second part of this section investigates and evaluates different onboard data processing pipelines. It looks into what is considered interesting onboard data processing pipelines and uses the two evaluation models discussed in this thesis to assess them. We divide the processing pipelines into five groups, the processing pipelines that aim to compress the data, do target detection or anomaly detection with hardware implementations, do target detections with software implementations, do anomaly detections with software implementations, or classify the data. First, we evaluate the groups separately, and then the most interesting pipelines from each group are compared.

We include discussion of the results as they arrive.

6.1 The Algorithms Run Times

In this section, we display the estimated run times of the algorithms on three hyperspectral cubes of different data sizes. In this section, we use operation count to estimate run time, where the assumed operating frequency is 667 MHz, as this is the operating frequency of HYPSO-1 [57].

The first hyperspectral cube consists of 956 frames, 684 frame samples, and 1080 spectral bands, the standard hyperspectral cube size of HYPSO-1 images.

The second cube equals the first cube but with binned spectral values. We let the spectral binning factor be nine, as this is the standard binning factor of HYPSO-1. Consequently, the second cube consists of only 120 bands. This hyperspectral cube is assumed to be a binned cube, and we are therefore not interested in binning it once more. For this cube input, we, therefore, discard the run times estimations of the binning algorithms.

In the last hyperspectral cube, we assume a dimensional reduction algorithm has reduced the spectral dimension to only 20 bands. By allowing spectral data to be represented by 20 bands, we ensure high data quality and a high reduction of the spectral domain. It is interesting for HYPSO to check how different algorithms run using a reduced spectral dimension domain.

For this dimensional reduced cube, it is only interesting to investigate the run times of geometric registration, georeferencing, the interpreting algorithms, and the CCSDS123 compression algorithms. The other algorithms cannot operate on spectral values that are reduced by dimensional reduction algorithms, or they are the dimensional reduction algorithms themselves. Binning is also excluded from the generated run times.

We are interested in these three cube sizes because HYPSO-1's standard raw cube is 956x684x1080, and HYPSO-1 has decided upon a spectral binning factor of nine. This means that it is pretty likely that processing algorithms will operate on one of these hyperspectral data sizes. In addition, HYPSO is interested in investigating how more dimensional reduction can enable even more onboard data processing.

In figure 6.1 each algorithm is assigned a number and a color, as which is displayed on the right side. The upper run time estimations are the run times when inputting a hyperspectral data cube of dimensions 956x684x1080. The middle run time estimations have a hyperspectral data input of 956x684x120, while the run time estimates displayed at the bottom of figure 6.1 assume an input of size 956x684x20.

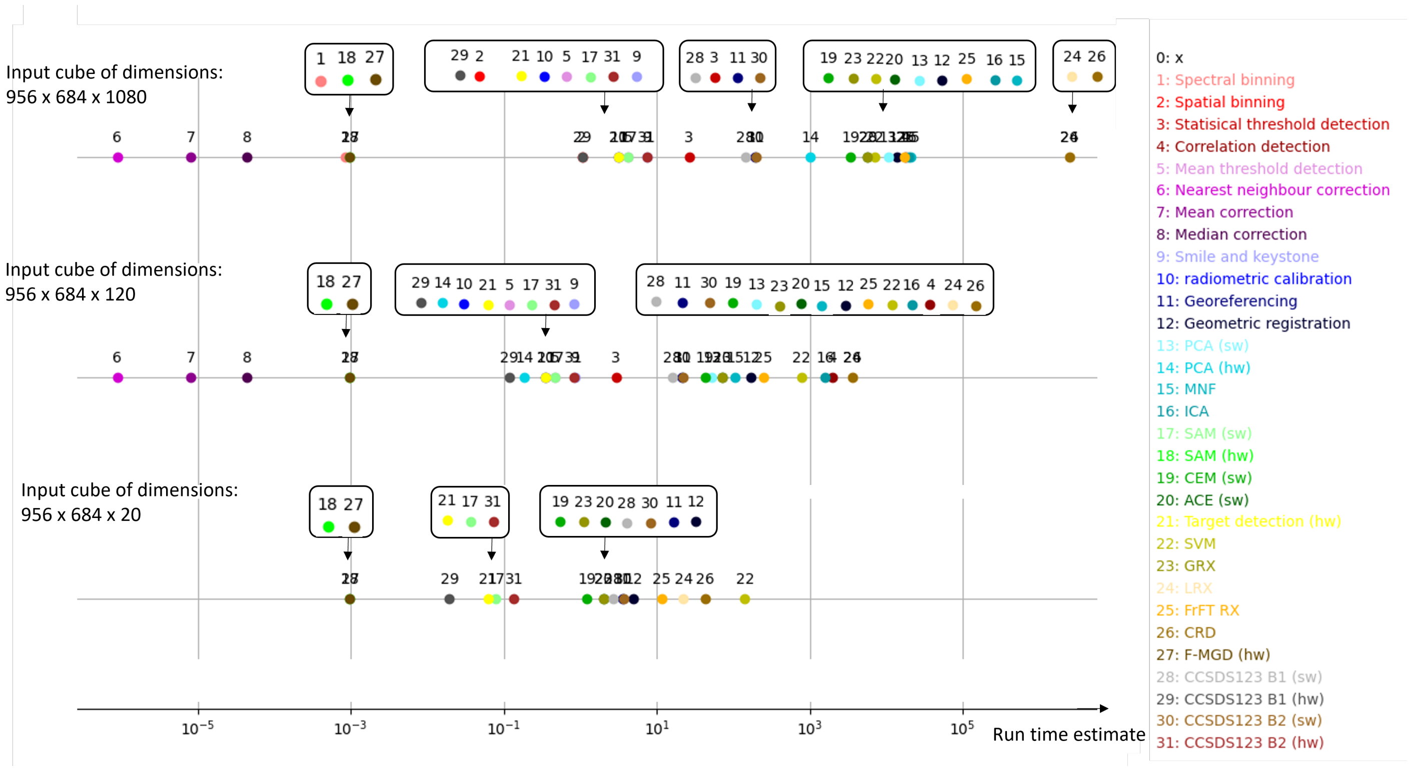


Figure 6.1: The run time estimates when inputting hyperspectral data cube of sizes 956x684x1080, 956x684x120, and 956x684x20.

By investigating the figure 6.1, we make the following observations.

1. The estimated run times of the hardware implementation of SAM and F-MGD are very fast and primarily dependent on the spatial dimensions. If we want to apply the hardware algorithms SAM or F-MGD on a hyperspectral image, it is probably most efficient to do so on a raw hyperspectral cube.
2. The estimated run time of spectral binning is much faster than spatial binning. The big difference in run time is probably due to spectral binning being implemented with SIMD instructions and spatial binning with only SISD instructions. Considering that spectral binning is superior to spatial binning in estimated run time and data quality, as elaborated in section 4.2.2.2.1, we should always prefer spectral binning over spatial binning.
3. The bad pixel detection algorithm, mean threshold detection, is much faster than the other bad pixel detection algorithms. Therefore, we should always use mean threshold detection when using a bad pixel detection algorithm.
4. The nearest neighbor correction algorithm is much faster than the other bad pixel correction algorithms. We, therefore, always use nearest neighbor correction when correcting bad pixels.

5. The hardware implementation of PCA is the fastest dimensional reduction algorithm. Compared to PCA implemented in hardware, the other dimensional reduction algorithms have a very high estimated run time. Consequently, using the hardware implementation of PCA saves a lot of processing time and is, therefore, the primary choice of dimensionality reduction algorithm in the onboard data processing pipelines.
6. The estimated processing times of geometric registration, the interpreting algorithms, and the CCSDS123 compression algorithms reduce much when decreasing the spectral dimension.
7. The hardware implementations are always better than the software implementations of the same algorithm. Therefore, if given a choice, we use the hardware implementations.

6.2 Evaluation of Onboard Data Processing Pipelines

This section uses the conclusion found in section 6.1 to generate interesting processing pipelines.

In the following subsection, we list various data processing pipelines and, at the same time, provide them with a number. We see these numbers in the two figures in the subsection that display different evaluations of various processing pipelines. The numbers represent the listed processing pipelines in the given subsection.

6.2.1 Compression

In this section, we look into certain processing pipelines that aim to compress data.

It is interesting to see what effect binning has when compressing the data using CCSDS123 algorithms. But knowing that the software versions are much slower than the hardware versions, we are more interested in the hardware versions.

We, therefore, evaluate the following processing pipelines.

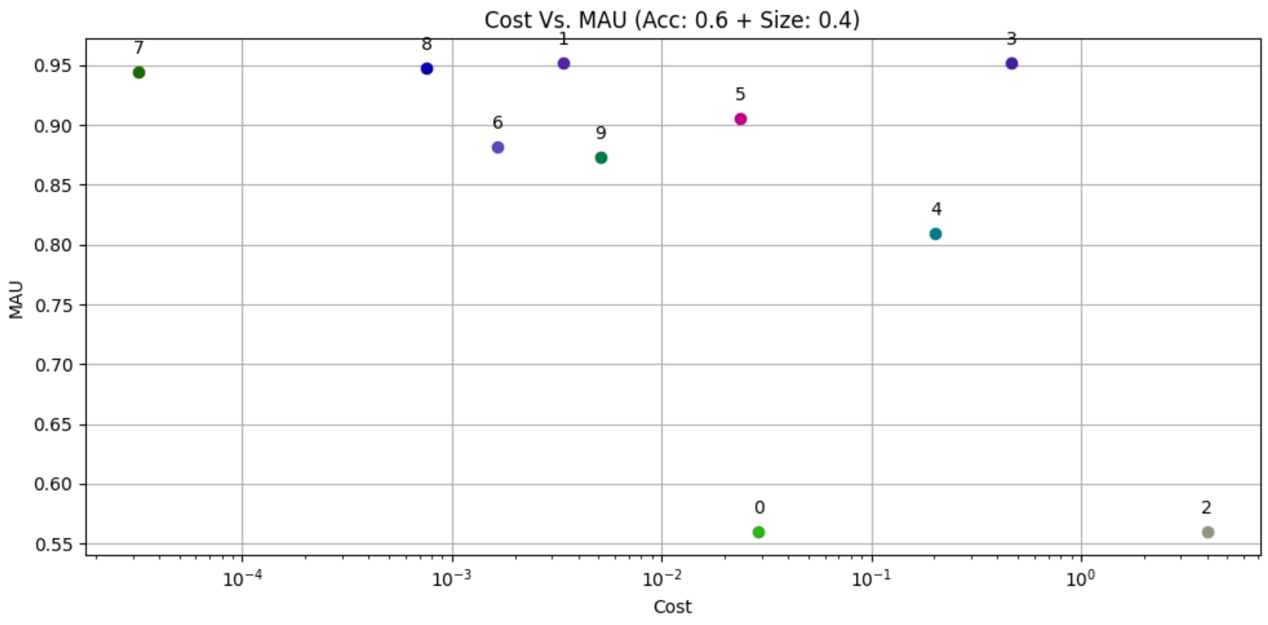
0. CCSDS123-B1 (hw)
1. Spectral binning, CCSDS123-B1 (hw)
2. CCSDS123-B1 (sw)
3. Spectral binning, CCSDS123-B1 (sw)
4. CCSDS123-B2 (hw)
5. Spectral binning, CCSDS123-B2 (hw)

It is also possible to compress data using dimensional reduction algorithms and evaluate processing pipelines holding PCA (hw). The following list of processing pipelines shows how PCA (hw) is affected by spectral binning and how we can reduce the data even more by adding CCSDS123 compression.

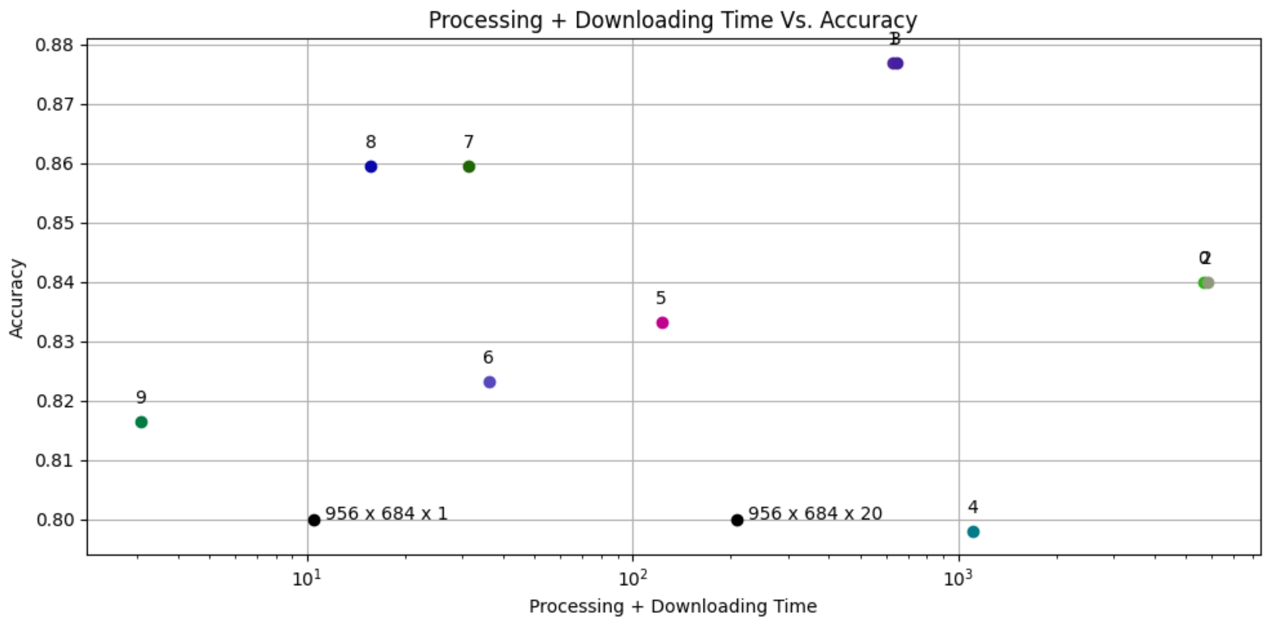
6. PCA (hw)
7. Spectral binning, PCA (hw)
8. Spectral binning, PCA (hw), CCSDS123-B1 (hw)
9. Spectral binning, PCA (hw), CCSDS123-B2 (hw)

It is extra interesting to check out the performance of the processing pipeline consisting of spectral binning, PCA (HW), and CCSDS123-B1 (hw). By downloading PCA's transformation matrix in addition to the data, it is possible to convert the reduced spectral dimension back to the original spectral dimensions on Earth. Suppose we can represent the data with the original spectral dimension at Earth. In that case, postprocessing can include correcting algorithms that require the spectral dimension to be represented in the original spectral components. An example of an algorithm that can be applied as a postprocessing algorithm in such cases is smile and keystone correction.

Figure 6.2a compares MAU score against cost, while figure 6.2b compares accuracy against processing and downloading time.



(a) MAU Vs. data processing time.



(b) Accuracy Vs. processing and downloading time. The black data points is reference data points that tells how much time is needed to download data of size 956 x 684 x 1 and 956 x 684 x 20.

Figure 6.2: Design space exploration when processing with compression algorithms.

Figure 6.2a shows that processing pipeline 7 provides the best benefit given the cost. Figure 6.2b also confirm that the time required to handle the processing chain of 7 is not that much and that the accuracy estimate is mediocre.

The evaluation models evaluate pipeline 8 quite well too. Processing pipeline 8 has more run time cost than 7, but will be faster to download to the Earth, as seen in figure 6.2b. Pipeline 9 is the pipeline that requires the least amount of time but does not provide good accuracy or MAU score.

According to this design space exploration, the best pipelines for compression are 7, 8, and 9.

6.2.2 Classification

For the classification algorithm, it is interesting to see how spectral binning, dimensionality reduction, and correcting algorithms affect the algorithm.

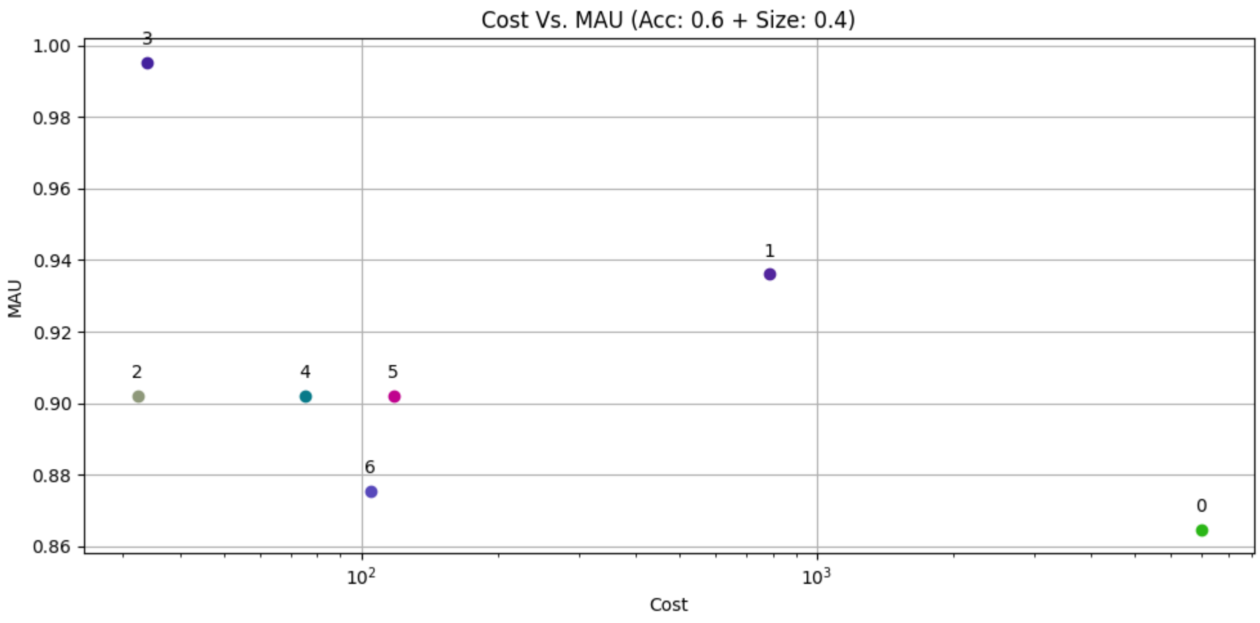
0. SVM
1. Spectral binning, SVM
2. Spectral binning, PCA (hw), SVM
3. Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, smile and keystone correction, radiometric calibration, PCA (hw), SVM

As it takes a lot of time to do SVM processing, we also check how much different dimensional reduction algorithms will affect the output.

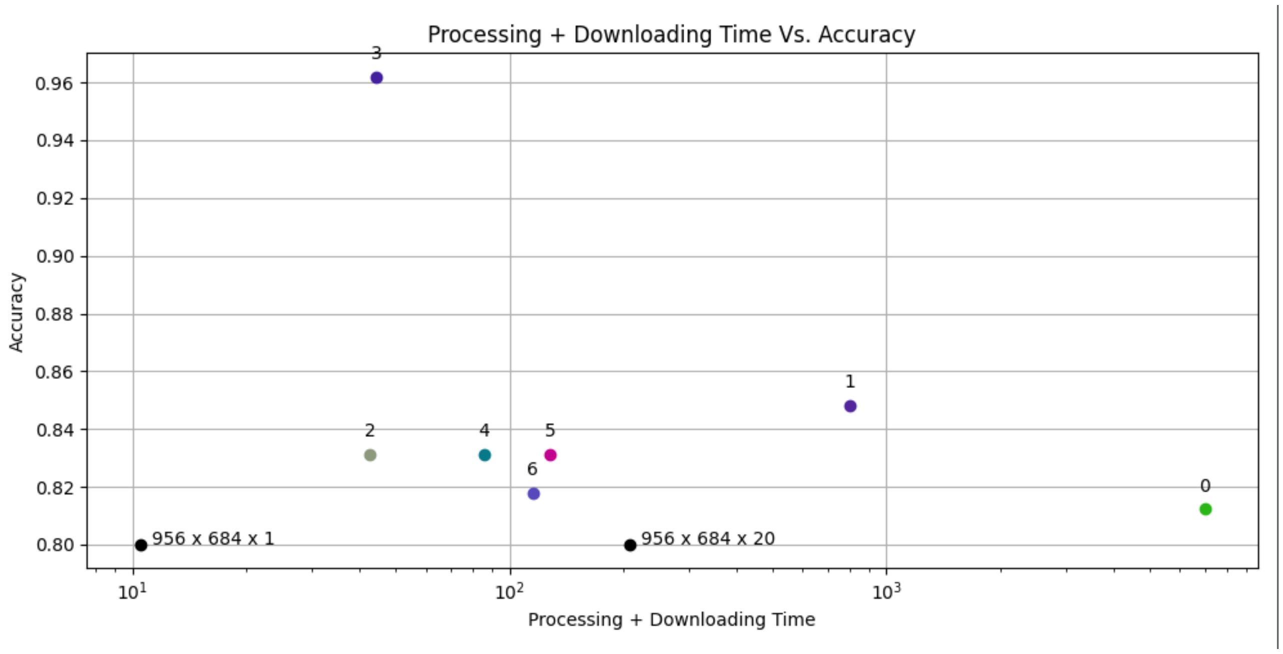
4. Spectral binning, PCA (sw), SVM
5. Spectral binning, MNF, SVM
6. Spectral binning, ICA, SVM

The output of SVM is a two dimensional image, and it is therefore not necessary to reduce the outputted data size more by CCSDS123 compression. The same goes for the pipelines that include target or anomaly detection algorithms.

The high run time estimate of the SVM algorithm is questionable. The run time estimate is not verified, and there is therefore a possibility that it is wrong. Consequently, we should not dismiss the classification algorithm based on the results of this design space exploration but rather do more research and strive to verify it. Figure 6.3 shows the interesting classification processing pipelines.



(a) MAU Vs. data processing time.



(b) Accuracy Vs. processing and downloading time. The black data points is reference data points that tells how much time is needed to download data of size 956 x 684 x 1 and 956 x 684 x 20.

Figure 6.3: Design space exploration when processing with a classification algorithm.

Pipeline 3 is the most interesting pipeline as it has a high MAU and accuracy score and low processing time compared to the other pipelines.

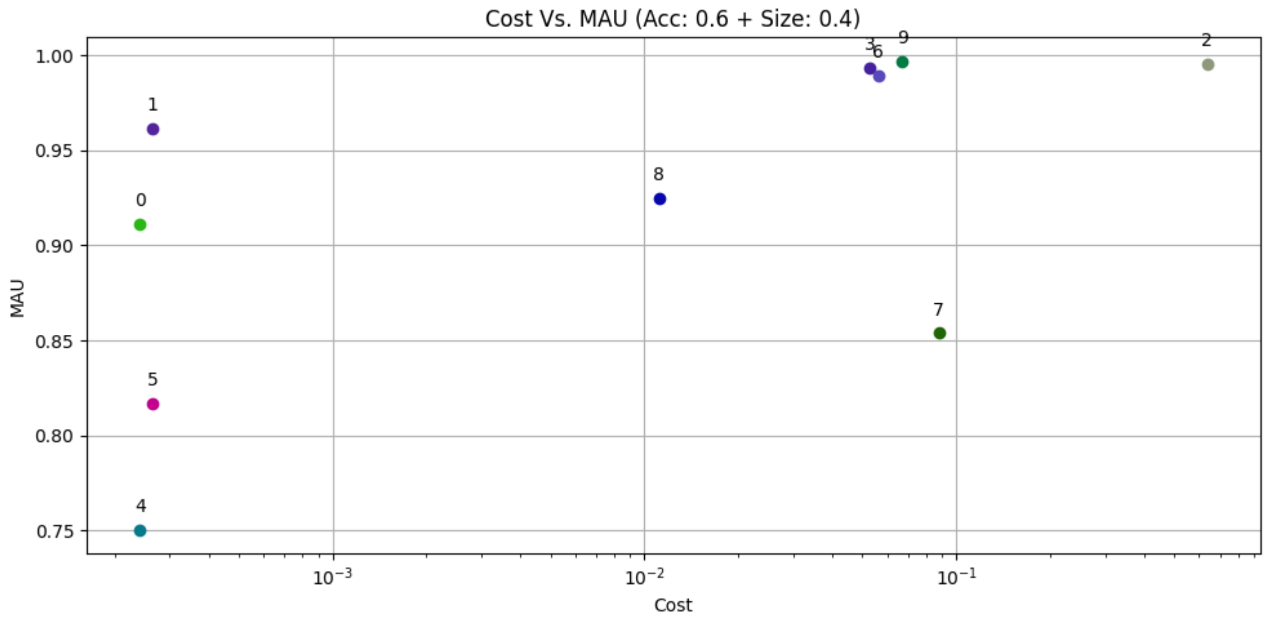
6.2.3 Hardware Implementations of Target Detection and Anomaly Detection

Hardware implementations of anomaly detection and target detection algorithms are relatively fast, and it is therefore interesting to see how spectral binning will affect them. The hardware implementation of the target detection algorithms CEM, ACE, and ASMF is equal, so we let CEM represent this target detection implementation.

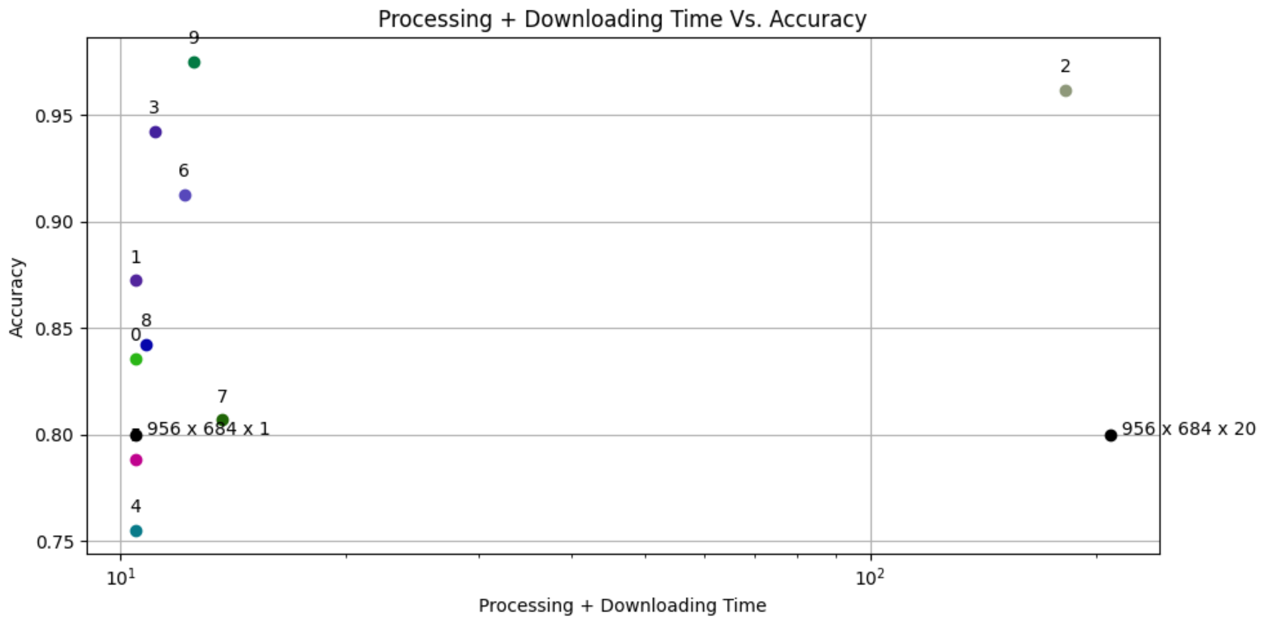
It is also interesting to see how correcting algorithms will affect the output. We add the same correcting algorithms for the target detection algorithms as for classification. We discard smile, keystone correction, and radiometric calibration for anomaly detection but add the geometric registration algorithm. Anomaly detection is not that sensitive for special spectral signatures but is sensitive for spatial shapes. The spatial shapes in the image should therefore imitate Earth as well as possible. The geometric registration algorithm can be applied after dimensional reduction, so it is interesting to see if dimensional reduction will positively or negatively affect the output.

We include the following processing pipelines:

0. F-MGD (hw)
1. Spectral binning, F-MGD (hw)
2. Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, geometric registration, F-MGD (hw)
3. Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, PCA (hw), geometric registration, F-MGD (hw)
4. SAM (hw)
5. Spectral binning, SAM (hw)
6. Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, smile and keystone correction, radiometric calibration, SAM (hw)
7. CEM (hw)
8. Spectral binning, CEM (hw)
9. Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, smile and keystone correction, radiometric calibration, CEM (hw)



(a) MAU Vs. data processing time.



(b) Accuracy Vs. processing and downloading time. The black data points is reference data points that tells how much time is needed to download data of size 956 x 684 x 1 and 956 x 684 x 20.

Figure 6.4: Design space exploration when processing with hardware implementations of target and anomaly detection algorithms.

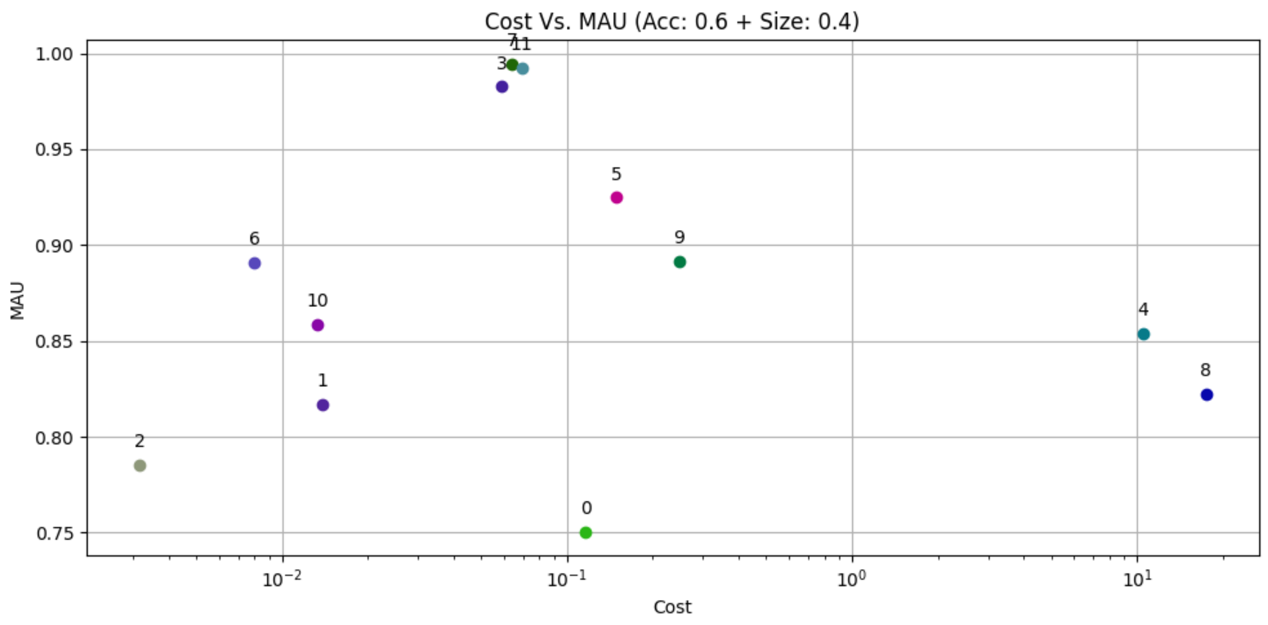
In figure 6.4b most of the pipelines have the same or almost the same total time usage. This is because the total time usage includes downloading the data. For these pipelines, downloading data is the most time consuming part of the data handling.

The most interesting pipeline is pipeline 9, because it has the highest MAU score and the highest accuracy estimate. Even though pipeline 9 has a high run time compared with other pipelines, the run time is still short, and the total time usage 9 requires is not that much more than the other pipelines.

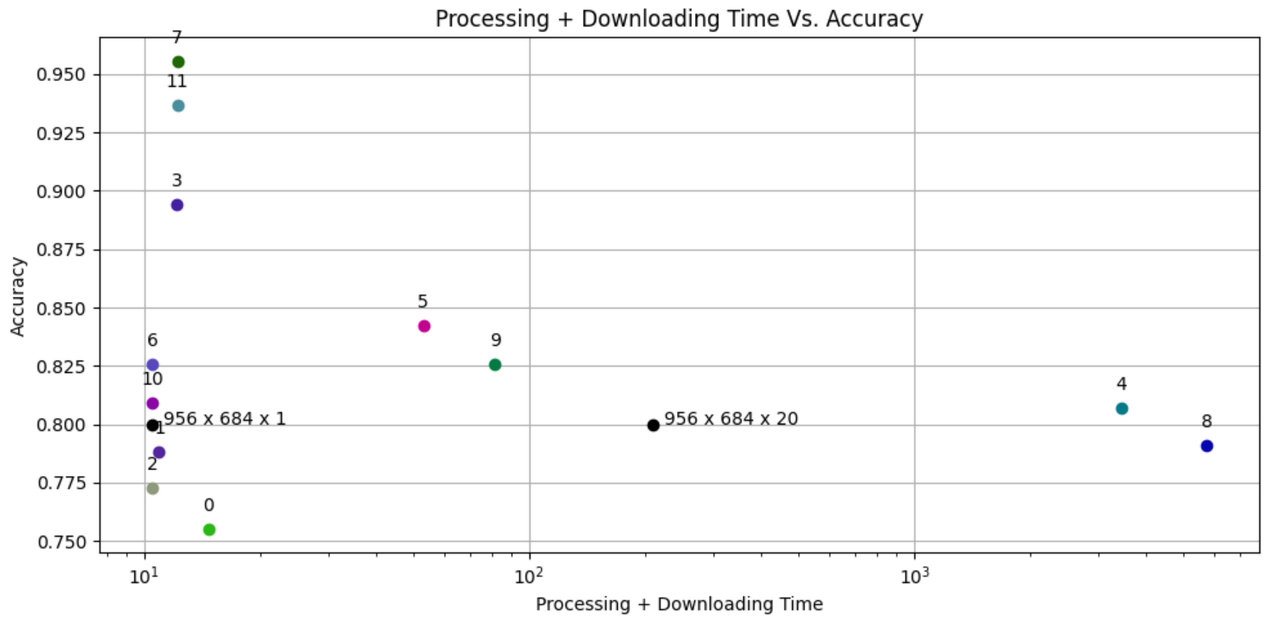
6.2.4 Software Implementations of Target Detection Algorithms

For the software implementations of target detection algorithms, it is interesting to see how spectral binning, dimensional reduction, and correcting algorithms affect the outputs. The data processing pipelines included are, therefore, the following.

0. SAM (sw)
1. Spectral binning, SAM (sw)
2. Spectral binning, PCA (hw), SAM (sw)
3. Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, smile and keystone correction, radiometric calibration, PCA (hw), SAM (sw) -
4. CEM (sw)
5. Spectral binning, CEM (sw)
6. Spectral binning, PCA (hw), CEM (sw)
7. Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, smile and keystone correction, radiometric calibration, PCA (hw), CEM (sw)
8. ACE (sw)
9. Spectral binning, ACE (sw)
10. Spectral binning, PCA (hw), ACE (sw)
11. Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, smile and keystone correction, radiometric calibration, PCA (hw), ACE (sw)



(a) MAU Vs. data processing time.



(b) Accuracy Vs. processing and downloading time. The black data points is reference data points that tells how much time is needed to download data of size 956 x 684 x 1 and 956 x 684 x 20.

Figure 6.5: Design space exploration when processing with software implementations of target detection algorithms.

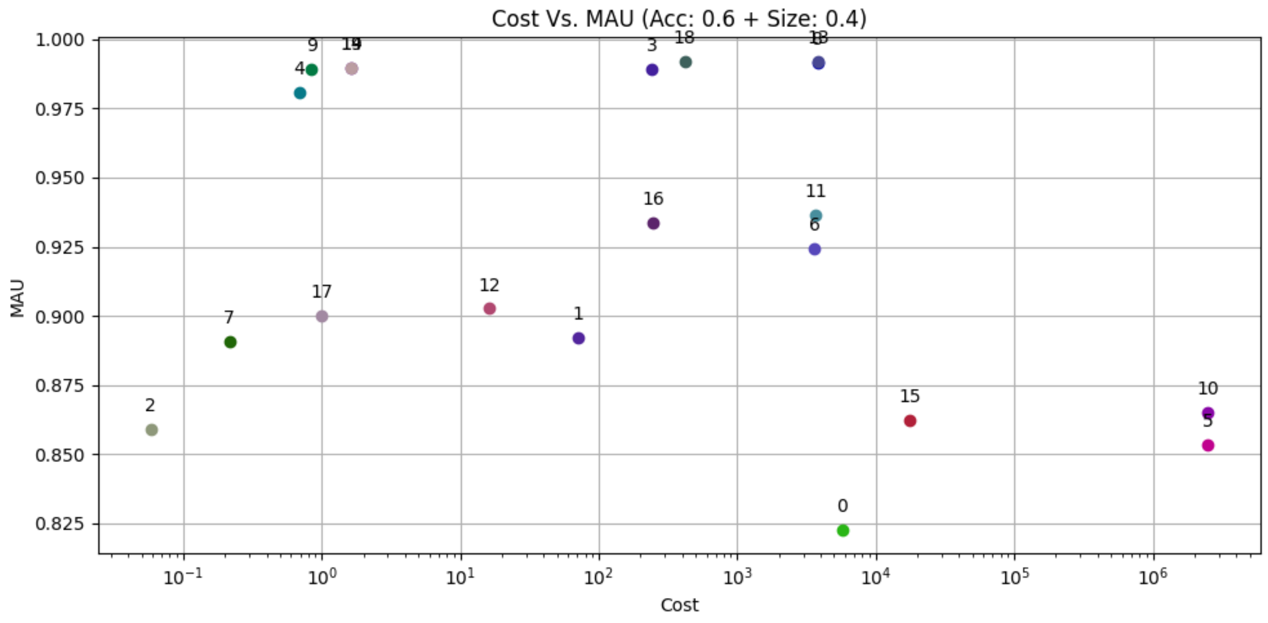
The most interesting pipeline is 7, even though 2 has the fastest processing time. 7 is very interesting as it has a high MAU and accuracy score, and even though it needs more processing time than 2, it is pretty fast. For both 7 and 2 the data downloading dominates the total time usage.

Other interesting pipelines are 11 and 3, as these also have high MAU and accuracy scores and low run time.

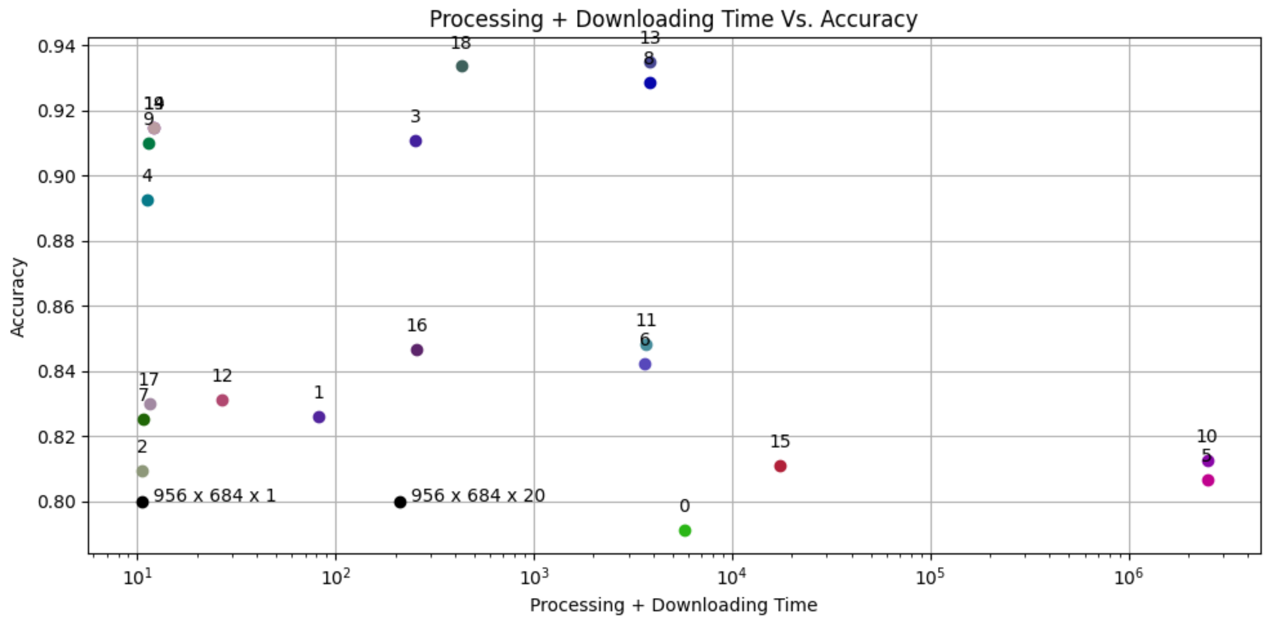
6.2.5 Software Implementations of Anomaly Detection Algorithms

The software implementations of the anomaly detection algorithms should also be asses as the target detection algorithms, with spectral binning, dimensional reduction, and correcting algorithms. The interesting data processing pipelines are the following.

0. GRX (sw)
1. Spectral binning, GRX (sw)
2. Spectral binning, PCA (hw), GRX (sw)
3. Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, geometric registration, GRX (sw)
4. Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, PCA (hw), geometric registration, GRX (sw)
5. LRX (sw)
6. Spectral binning, LRX (sw)
7. Spectral binning, PCA (hw) LRX (sw)
8. Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, geometric registration, LRX (sw)
9. Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, PCA (hw), geometric registration, LRX (sw)
10. CRD (sw)
11. Spectral binning, CRD (sw)
12. Spectral binning, PCA (hw) CRD (sw)
13. Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, geometric registration, CRD (sw)
14. Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, PCA (hw), geometric registration, CRD (sw)
15. FrFT-RX (sw)
16. Spectral binning, FrFT-RX (sw)
17. Spectral binning, PCA (hw), FrFT-RX (sw)
18. Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, geometric registration, FrFT-RX (sw)
19. Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, PCA (hw), geometric registration, FrFT-RX (sw)



(a) MAU Vs. data processing time.



(b) Accuracy Vs. processing and downloading time. The black data points is reference data points that tells how much time is needed to download data of size 956 x 684 x 1 and 956 x 684 x 20.

Figure 6.6: Design space exploration when processing with software implementations of anomaly detection algorithms.

The three most interesting pipelines are 19, 9, and 4. These pipelines have a high MAU score and a relatively low processing time. The processing time is so small that most of the time used to handle the pipelines is used in data downloading. Therefore it is still preferable to use pipelines 19, 9, or 4 compared to 2, even though 2 needs less processing time.

6.2.6 Overview of the Most Interesting Onboard Data Processing Pipelines

We have evaluated different kinds of onboard data processing pipelines in the above subsections. The following list provides an overview of which processing pipelines we considered the most interesting.

- Spectral binning, PCA (hw)
- Spectral binning, PCA (hw), CCSDS123-B1 (hw)
- Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, smile and keystone correction, radiometric calibration, PCA (hw), SVM
- Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, smile and keystone correction, radiometric calibration, CEM (hw)
- Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, smile and keystone correction, radiometric calibration, PCA (hw), CEM (sw)
- Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, smile and keystone correction, radiometric calibration, PCA (hw), ACE (sw)
- Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, smile and keystone correction, radiometric calibration, PCA (hw), SAM (sw)
- Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, PCA (hw), geometric registration, FrFT-RX (sw)
- Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, PCA (hw), geometric registration, LRX (sw)
- Spectral binning, bad pixel mean threshold detection, bad pixel nearest neighbor correction, PCA (hw), geometric registration, GRX (sw)

From the following list, we can conclude that it is beneficial to add correcting algorithms to the processing pipelines and that compression using PCA (hw) looks promising.

Chapter 7

Conclusion and Further Work

In this master thesis, we have conducted a design space exploration of onboard data processing pipelines. This is made in cooperation with a NTNU team called HYPSO, which aims to use hyperspectral images to detect poisonus algee blooms. The hyperspectral images are taken from space by using satellites. The results of the design space exploarion should help the HYPSO team plan for further research. The HYPSO team can get an insight into which algorithms to include in an onboard data processing pipeline potentially.

It is important to note that this design space exploration uses rough accuracy and run time estimations. Consequently, the results from this design space exploration should not be used as facts. As stated in the background section and shown in the verification section, the run times estimates of the algorithms are very optimistic and unrealistic.

According to this design space exploration, the most promising onboard data processing pipelines include correcting algorithms. Compression by dimensional reduction is also shown to be a good processing pipeline. If we use dimensionality reduction as compression and download the dimensionality reduction's transformation matrix, it can be possible to apply algorithmic corrections as a postprocessing step on Earth. This can save onboard data processing time and increase the data quality.

Most of the outputted data size estimations are deterministic and easily understood; therefore, we can be pretty confident that these estimates are correct. The accuracy and run time estimates, on the other hand, are more likely to be a bit off. The accuracy estimate is based on several facts about the data, and the run time estimates exclude all kinds of operational overhead. However, by using the same estimation strategies for all the processing pipelines, we can at least assume the estimates are wrong in a consistent manner. Consequently, it should be possible to compare the processing pipelines based on the estimates.

Sadly, there is only a limited amount of onboard processing pipelines that are displayed and discussed in this thesis. According to the author this thesis display the most interesting pipeline combinations, but other might disagree. The processing pipelines have assumed that the interesting data sizes are those used in the HYPSO-1 mission. These sizes can be irrelevant for other space missions, which also need onboard data processing. Lucky, this design space exploration in this thesis should be described with enough details which should make it possible to copy the work, and thereby checking out other data dimensions or other data processing pipelines.

For Further work, it would be interesting to expand the design space exploration by including more algorithms.

It would also be interesting to investigate the different run time estimations mentioned in the background section. Maybe it is sufficient to do run time estimation by using complexity analysis? Or would complexity analysis provide enough detailed information about the different algorithms? These are an interesting questions as complexity analysis can make a design space exploration much faster or more extensive.

Bibliography

[1] M. Eismann, *Hyperspectral Remote Sensing*. SPIE, 2012.

[2] M. E. Grøtte et al., 'Mission design, operations scheduler and payload trade-offs for hsi v6, hypso-1,' 2020.

[3] M. E. Grøtte et al., 'Ocean Color Hyperspectral Remote Sensing With High Resolution and Low Latency—The HYPSO-1 CubeSat Mission,' *IEEE Trans. Geosci. Remote Sens.*, pp. 1–19, 2021. DOI: `10.1109/TGRS.2021.3080175`.

[4] F. A. Agelet et. al., 'Ttt21 satellite systems engineering,' 2021.

[5] 'Mit se course 4.' (2022).

[6] G. D.D. et. al., *Specification and Design of Embedded Systems*. Prentice Hall, 1994, p. 234.

[7] P. G. Kjeldsberg, 'Hardware/software codesign why, what, and how?,' 2021.

[8] *Hyspex baldur v-1024 n*, V-1024 N, HySpex, Jul. 2022. [Online]. Available: `https://www.hyspex.com/media/aighw12k/hyspex_baldur_v-1024-n.pdf`.

[9] *Specim afx10*, AXF10, SPECIM, Jul. 2022. [Online]. Available: `https://www.specim.fi/wp-content/uploads/2020/02/Specim-AFX10-Technical-Datasheet-02.pdf`.

[10] M. D. Lewis, R. W. Gould, R. A. Arnone, P. E. Lyon, P. M. Martinolich, R. Vaughan, A. Lawson, T. Scardino, W. Hou, W. Snyder, R. Lucke, M. Corson, M. Montes and C. Davis, 'The hyperspectral imager for the coastal ocean (hico): Sensor and data processing overview,' in *OCEANS 2009*, 2009, pp. 1–9. DOI: `10.23919/OCEANS.2009.5422336`.

[11] B. A. Bradley, 'Remote detection of invasive plants: A review of spectral, textural and phenological approaches,' 2013.

[12] D. Keith, 'Coastal and estuarine waters: Optical sensors and remote sensing for management,' 2014. DOI: `10.1201/9780429441004-5`.

[13] M. Seelye, *An Introduction to Ocean Remote Sensing*. Cambridge University Press, 2014.

[14] C. João M.P. et. al., *Embedded Computing for High Performance Efficient Mapping of Computations Using Customization, Code Transformations and Compilation*. Elsevier, 2017, pp. 255–256.

[15] A. Ross et. al., 'Aligning perspectives and methods for value-driven design,' 2010.

[16] D. Von Winterfeldt and G. W. Fischer, 'Multi-attribute utility theory: Models and assessment procedures,' in *Utility, Probability, and Human Decision Making: Selected Proceedings of an Interdisciplinary Research Conference, Rome, 3–6 September, 1973*, D. Wendt and C. Vlek, Eds. Dordrecht: Springer Netherlands, 1975, pp. 47–85, ISBN: 978-94-010-1834-0. DOI: `10.1007/978-94-010-1834-0_3`. [Online]. Available: `https://doi.org/10.1007/978-94-010-1834-0_3`.

[17] J. D. Ullman, *Foundations of Computer Science: C Edition*. W. H. Freeman, 1994, pp. 92–97.

[18] Đ. Bošković et. al., 'A reconfigurable multi-mode implementation of hyperspectral target detection algorithms,' 2020.

[19] A. L. Gundersen, 'Hardware-software partitioned implementation of an autoencoder-based hyperspectral anomaly detector,' 2021.

[20] S. Narkhede. 'Understanding auc - roc curve.' (), [Online]. Available: https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5 (visited on 02/07/2022).

[21] G. Zeng et. al., 'A necessary condition for a good binning algorithm in credit scoring,' 2014.

[22] T. Koontz, 'Bad pixel detection and correction algorithms,' 2021.

[23] N. Yokoya et. al., 'Preprocessing of hyperspectral imagery with consideration of smile and keystone properties,' 2010. DOI: 10.1117/12.870437.

[24] M. B. Henriksen, 'Hyperspectral imager calibration and image correction,' 2019.

[25] E. Ibarrola, J. Marcello-Ruiz and C. Gonzalo-Martin, 'Evaluation of dimensionality reduction techniques in hyperspectral imagery and their application for the classification of terrestrial ecosystems,' Oct. 2017, p. 17. DOI: 10.1117/12.2278501.

[26] S. Bakken et. al., 'The effect of dimensionality reduction on signature based target detection for hyperspectral remote sensing,' 2019.

[27] C.-I. Chang and Q. Du, 'Interference and noise-adjusted principal components analysis,' *IEEE Transactions on Geoscience and Remote Sensing*, vol. 37, no. 5, pp. 2387–2396, 1999. DOI: 10.1109/36.789637.

[28] A. Tharwat, 'Independent component analysis: An introduction,' 2018. DOI: 10.1016/j.aci.2018.08.006.

[29] D. D. Langer, 'Image registration and georeferencing with snapshot camera for the hypso mission,' 2019.

[30] A. L. Gundersen, 'Hardware-software partitioned implementation of an autoencoder-based hyperspectral anomaly detector,' 2021.

[31] R. Tao et. al., 'Hyperspectral anomaly detection by fractional fourier entropy,' 2019.

[32] W. Li et. al., 'Collaborative representation for hyperspectral anomaly detection,' 2014. DOI: 10.1109/TGRS.2014.2343955.

[33] J. Lei, G. Yang, W. Xie, Y. Li and X. Jia, 'A low-complexity hyperspectral anomaly detection algorithm and its fpga implementation,' *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. PP, pp. 1–1, Oct. 2020. DOI: 10.1109/JSTARS.2020.3034060.

[34] F. M. Riese, S. Keller and S. Hinz, 'Supervised and semi-supervised self-organizing maps for regression and classification focusing on hyperspectral data,' *Remote Sensing*, vol. 12, no. 1, 2020, ISSN: 2072-4292. DOI: 10.3390/rs12010007. [Online]. Available: https://www.mdpi.com/2072-4292/12/1/7.

[35] F. Melgani and L. Bruzzone, 'Classification of hyperspectral remote sensing images with support vector machines,' *IEEE Transactions on Geoscience and Remote Sensing*, vol. 42, no. 8, pp. 1778–1790, 2004. DOI: 10.1109/TGRS.2004.831865.

[36] R. Gandhi. 'Support vector machine — introduction to machine learning algorithms.' (), [Online]. Available: https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47 (visited on 05/07/2022).

[37] C. Boothby, 'An implementation of a compression algorithm for hyperspectral images. a novelty of the ccsds 123.0-b-2 standard, master thesis,' 2020.

[38] E. F. Prentice, M. E. Grøtte, F. Sigernes and T. A. Johansen, 'Design of a hyperspectral imager using COTS optics for small satellite applications,' in *International Conference on Space Optics — ICSO 2020*, B. Cugny, Z. Sodnik and N. Karafolas, Eds., International Society for Optics and Photonics, vol. 11852, SPIE, 2021, pp. 2154–2171. DOI: `10.1117/12.2599937`. [Online]. Available: `https://doi.org/10.1117/12.2599937`.

[39] S. U. Rehman, A. Kumar and A. Banerjee, 'SNR improvement for hyperspectral application using frame and pixel binning,' in *Earth Observing Missions and Sensors: Development, Implementation, and Characterization IV*, X. J. Xiong, S. A. Kuriakose and T. Kimura, Eds., International Society for Optics and Photonics, vol. 9881, SPIE, 2016, pp. 134–139. DOI: `10.1117/12.2220599`. [Online]. Available: `https://doi.org/10.1117/12.2220599`.

[40] D. Fernandez, C. Gonzalez, D. Mozos and S. Lopez, 'Fpga implementation of the principal component analysis algorithm for dimensionality reduction of hyperspectral images,' vol. 16, no. 5, pp. 1395–1406, Oct. 2019, ISSN: 1861-8200. DOI: `10.1007/s11554-016-0650-7`. [Online]. Available: `https://doi.org/10.1007/s11554-016-0650-7`.

[41] S. Leavesley, B. Sweat, C. Abbott, P. Favreau and T. Rich, 'A theoretical-experimental methodology for assessing the sensitivity of biomedical spectral imaging platforms, assays, and analysis methods,' *Journal of biophotonics*, vol. 11, May 2017. DOI: `10.1002/jbio.201600227`.

[42] D. Boskovic, 'Hardware implementation of a target detection algorithm for hyperspectral images,' 2019.

[43] X. Kang, X. Zhang, S. Li, K. Li, J. Li and J. A. Benediktsson, 'Hyperspectral anomaly detection with attribute and edge-preserving filters,' *IEEE Transactions on Geoscience and Remote Sensing*, vol. 55, no. 10, pp. 5600–5611, 2017. DOI: `10.1109/TGRS.2017.2710145`.

[44] F. Andika, M. Rizkinia and M. Okuda, 'A hyperspectral anomaly detection algorithm based on morphological profile and attribute filter with band selection and automatic determination of maximum area,' *Remote Sensing*, vol. 12, no. 20, 2020, ISSN: 2072-4292. DOI: `10.3390/rs12203387`. [Online]. Available: `https://www.mdpi.com/2072-4292/12/20/3387`.

[45] C. Gonzalez et. al., 'Fpga implementation of the hysime algorithm for the determination of the number of endmembers in hyperspectral data,' *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 6, pp. 2870–2883, 2015. DOI: `10.1109/JSTARS.2015.2425731`.

[46] 'Matrix multiplication.' (), [Online]. Available: `https://en.wikipedia.org/wiki/Matrix_multiplication` (visited on 14/06/2022).

[47] C. Gonzalez, S. Lopez, D. Mozos and R. Sarmiento, 'Fpga implementation of the hysime algorithm for the determination of the number of endmembers in hyperspectral data,' *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 8, no. 6, pp. 2870–2883, 2015. DOI: `10.1109/JSTARS.2015.2425731`.

[48] C. Gonzalez, S. Lopez, D. Mozos and R. Sarmiento, 'A novel fpga-based architecture for the estimation of the virtual dimensionality in remotely sensed hyperspectral images,' *Journal of Real-Time Image Processing*, vol. 15, Jan. 2015. DOI: `10.1007/s11554-014-0482-2`.

[49] 'Sorting algorithms.' (), [Online]. Available: `https://brilliant.org/wiki/sorting-algorithms/` (visited on 14/06/2022).

[50] 'Merge sort.' (), [Online]. Available: `https://brilliant.org/wiki/merge/` (visited on 14/06/2022).

[51] NTNU SmallSat Lab, *Source code repository on github for "hypso-sw"*, [Online]. Available: https://github.com/NTNU-SmallSat-Lab/hypso-sw.

[52] J. Fjeldtvedt, M. Orlandić and T. A. Johansen, 'An efficient real-time fpga implementation of the ccsds-123 compression standard for hyperspectral images,' *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 11, no. 10, pp. 3841–3852, 2018. DOI: 10.1109/JSTARS.2018.2869697.

[53] Y. Barrios et. al, 'Hardware implementation of the ccsds 123.0-b-2 near-lossless compression standard following an hls design methodology,' *Remote Sensing*, vol. 13, no. 21, 2021. DOI: 10.3390/rs13214388.

[54] NTNU SmallSat Lab, *Source code repository on github for "onboard-pipeline-modules"*, [Online]. Available: https://github.com/NTNU-SmallSat-Lab/onboard-pipeline-modules.

[55] A. Bjorgan and L. L. Randeberg, 'Real-time noise removal for line-scanning hyperspectral devices using a minimum noise fraction-based approach,' *Sensors*, vol. 15, no. 2, pp. 3362–3378, 2015, ISSN: 1424-8220. DOI: 10.3390/s150203362. [Online]. Available: https://www.mdpi.com/1424-8220/15/2/3362.

[56] C. Maklin. 'Independent component analysis (ica) in python.' (2019), [Online]. Available: https://towardsdatascience.com/independent-component-analysis-ica-in-python-a0ef0db0955e (visited on 09/06/2022).

[57] A. Varntresk, 'Assembly and testing of baseline processing chain,' 2019.

Kristine Døsvik

NTNU
Norwegian University of
Science and Technology