Technical Report

Pattern Recognition and Image Processing Group
Institute of Computer Aided Automation
Vienna University of Technology
Favoritenstr. 9/183-2
A-1040 Vienna AUSTRIA
Phone: +43 (1) 58801-18366
Fax: +43 (1) 58801-18392
E-mail: nabil@prip.tuwien.ac.at
URL: http://www.prip.tuwien.ac.at/

PRIP-TR-088 April 13, 2004

On-Ground Astronomical Data Processing
“A Software Tool for HERSCHEL/PACS Data Decompression and Analysis”

Ernst Hirz, Ahmed Nabil Belbachir and Robert Sablatnig

Abstract

This work is about the design and development of a standalone tool for the decompression of data from the **P**hotodetector **A**rray **C**amera and **S**pectrometer (PACS), one of the instruments housed inside the **H**ERSCHEL **S**pace **O**bservatory (HSO). This is a part of an on-ground processing software package, with the purpose to provide an instrument for collecting, assembling, decompressing and later analyzing the data fragments received from the telescope. The work is done in JAVA programming language for simplicity, portability, reliability and a distributed computing. Furthermore, JAVA is object-oriented. Object-oriented programming provides greater flexibility, modularity and reusability. Thus it is easy to maintain/extend the on-ground processing tool for different or further compression algorithms or general upgrades. Within this work, we developed a scientific software tool for the processing of the received data from the HSO/PACS instrument. This software tool is designed and tested for PACS data but not limited for astronomical applications. The on-ground processing concept can easily be adapted for other applications (Medical imaging, Telecommunication…etc). Results from the evaluation of the software with real and virtual test data are given at the end of this report.

Contents

1 Introduction

A new form of communication has evolved over the last two decades. This involves mobile communication, the ever growing internet and video communication for example. Data compression is one of the enabling technologies for this multimedia revolution [11]. That also applies for astronomy applications, where data is collected on-board a spacecraft [18].
Optical and radar imaging sensors together with their specific characteristics and their common user requirements is intended as a basis for typical spaceborne imaging applications, where high-rate sensors deliver more data than can be transmitted directly to ground [4]. **I**nfra**R**ed (IR) detectors consist of fewer pixels than those for visual range [24], but the design of multi-sensor instruments leads to even higher data volumes [17]. Figure 1 shows an example of an IR image of the astronomy object Spiral galaxy M81 in three different wave-lengths. These images were taken with Spitzer Space Telescope [23].

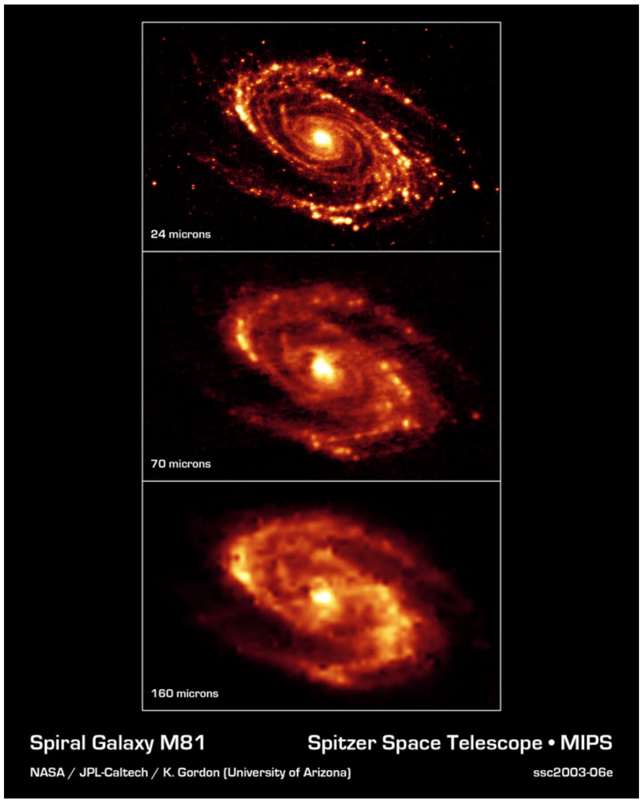
24 microns
70 microns
160 microns
Spiral Galaxy M81
Spitzer Space Telescope • MIPS
NASA / JPL-Caltech / K. Gordon (University of Arizona)
ssc2003-06e

Figure 1: IR-Image from Spitzer Space Telescope

If multiple detectors are operated in parallel to support multi-spectral or even hyper-spectral imaging, the data volumes multiply [17]. Transmitting image information face a bottleneck such, that this constraint has stimulated advances in compression techniques for astronomy [19]. Within the compression/decompression chain, the ground segment does not only serve as a facility for decompression of transmitted image data but can make use of much more advanced functions. These functions may include error concealment during compression (i.e. hiding of decompression failures due to data loss during transmission), combined decompression and restoration (e.g. removal of compression and/or sensor effects), or combined compression and feature detection. State of the art in IR data analysis is the analysis

package for **I**nfrared **S**pace **O**bservatory (ISO) [21] and also the one for **I**nfra**R**ed **A**stronomical **S**atellite (IRAS) [22]. These packages were programmed in **I**nteractive **D**ata **L**anguage (IDL). Hence they are not programmed in a modular way it is more costly to adjust them for our need than to develop a new tool for the task. Promising initial results on data restoration and denoising using both wavelet and multiresolution-based on the median transform have been shown for ISO [20]. We intend to take advantage of these results for processing PACS instrument data within a standalone-processing environment for quick-look analysis and assessment.
In this report we focus on the decompression of HSO/PACS data. The objective of this work is to develop a software tool for the processing of the received data from the Telescope. This software tool is programmed in a modular object-oriented way.

Generally the task of the on-ground software can be subdivided into three major parts:

- Collection of the packets: The data processed by the telescope will be divided into fragments for transmission to earth. On-ground the compressed data needs to be regained from these data packets. This involves testing for transmission errors.
- Decompression of the data: The reconstructed compressed data blocks are decompressed with the reversals of the compression algorithms used on-board. This task includes correct handling of the data in case of errors. The user is supplied with information about the decompression run. Possibility to test the results of the decompression is given as well as computation of reconstruction error measures, which are explained in Section 2.2.
- Analysis of the reconstructed images: Once the original raw decompressed data is regained, the real images have to be recomputed and analysed. This includes tools for repairing images in case of transmission errors and evaluation of control and additional data.

The last item of the above summary of the on-ground software is not part of this work and will not be described in this work. Further reports about the on-ground software tool for the HERSCHEL/PACS Telescope will deal with this task.

This document consists of four main parts. First of all we discuss some basic stuff of data compression. The next section deals with the compression/decompression concept of the project. After that, this report is about the design and implementation of software, which one can use for collection and decompression of information, packed into data fragments. In this project this task was done specifically for the application of the HERSCHEL/PACS telescope. In section five we talk about the results of the evaluation of the implemented piece of software.

2 Data compression

Data compression is used to reduce the amount of bits required to represent information, for example an image. In this section we want to explain fundamental things about data compression. This includes methods to compute reconstruction error measures which can be used to evaluate compression techniques.

2.1 Basics about Data Compression

In brief, data compression is the technique to encode data in a way that it allows compact storage or efficient transmission of information. This is possible because real world data is redundant. In information theory, redundancy is the number of bits used to represent a message minus the number of bits of actual information in the message. Figure 2 shows a basic data compression/decompression block diagram. Original data gets compressed which yields compressed data. This compressed data can be used for storage or transmission. The right part of the diagram shows the decompression which result is the reconstructed data.

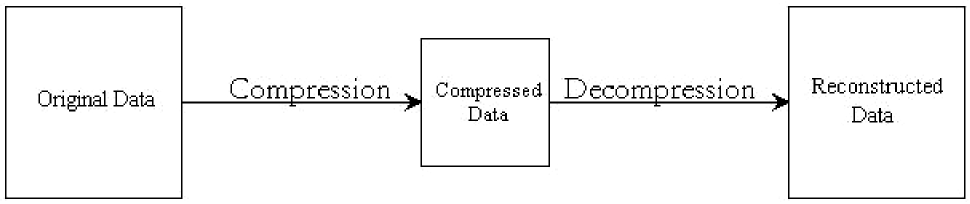

Figure 2: Data compression block diagram

To measure how well a compression algorithm compresses a given set of data we look at the amount of bits required to represent the data before and after compression. Let S be the original data and S_C the compressed data. The compression ratio is defined as follows:

$$Compression\ Ratio(CR) = \frac{S}{S_C}$$

Basically data compression methods can be categorized into two types – lossless or lossy compression. K. Sayood gives the following definition of these methods [11].

- Lossless compression techniques, as their name implies, involve no loss of information. If data have been losslessly compressed, the original data can be recovered exactly from the compressed data. Lossless compression is genereally used for applications that cannot tolerate any difference between the original and reconstructed data. Text compression is an important area for lossless compression.

- Lossy compression techniques on the other hand involve some loss of information, and data that have been compressed using lossy techniques generally cannot be recovered or reconstructed exactly. In return for accepting this distortion in the reconstruction, we can generally obtain much higher compression ratios than is possible with lossless compression. In many applications, this lack of exact reconstruction is not a problem. For example, when storing or transmitting speech, the exact value of each sample of speech is not necessary.

Compression methods are evaluated with different parameters. These are mainly, the time it takes to perform the algorithm on a given system, how much memory it uses and the reconstruction error versus the compression ratio achieved. The following section defines the term reconstruction error in this context.

2.2 Reconstruction Error

The reconstruction error is a parameter of the compression method to quantify the loss of information during the computation. Generally it is a computed value to describe the difference between the original and the reconstructed data. It can be used to measure the loss of data during a lossy compression, or to describe the loss of data due to transmission errors. There are different methods to compute reconstruction error measures. The methods vary in their way to describe the deviation of the data. Some of these measures are described in the following and can also be found in [10]. Let S be the original data, S_R the reconstructed data and N the absolute length of the data.

- **M**ean **A**bsolute **E**rror (MAE): This is the simplest measure, which computes the average absolute error and is defined as follows:

$$MAE(S,S_R)=\frac{1}{N}\sum_{x=1}^{N}\left|S(x)-S_R(x)\right|$$

- **R**oot **M**ean **S**quare **E**rror (RMSE): This measure takes the average of the squares of the absolute difference and takes the square root of the result. It’s calculated as the standard deviation of the reconstructed data relative to the original data.

$$RMSE(S,S_R)=\sqrt{\frac{1}{N}\sum_{x=1}^{N}\left(S(x)-S_R(x)\right)^2}$$

- **S**ignal to **N**oise **R**atio (SNR): The SNR measure includes the intensity of the original data by dividing it with the sum of squares of the absolute difference. It is a term for the ratio between the maximum meaningful signal and the background noise. The range of values is reduced by taking the logarithm.

$$SNR(S,S_R)=10\log_{10}\frac{\sum_{x=1}^{N}\left(S(x)\right)^2}{\sum_{x=1}^{N}\left(S(x)-S_R(x)\right)^2}$$

- **P**eak **S**ignal to **N**oise **R**atio (PSNR): The PSNR is the most commonly used reconstruction error measure for compression methods. The PSNR is defined as follows:

$$PSNR(S,S_R) = 10\log_{10}\frac{\left(\max(S(x)) - \min(S(x))\right)^2}{\frac{1}{N}\sum_{x=1}^{N}\left(S(x) - S_R(x)\right)^2}$$

These metrics help to quantify the loss of data due to an applied compression computation. For more information on these measures and different measures see also [2,13].

3 Compression/Decompression Concept

The whole project is about the development of a new space telescope – the HERSCHEL/PACS telescope. This is a space observatory covering the full far-infrared and submillimetre waveband. It will be located 1.5 million kilometres away from earth. A mirror will collect the light from distant and poorly known objects, such as newborn galaxies, and will focus it onto three instruments [1]. The data will be compressed on a digital signal processor based on the TSC21020E architecture for space application. An undetailed basic block diagram of the system is shown in Figure 3. The upper part of this diagram shows the processing of the data on-board the spacecraft. The lower part shows the reconstruction of the data on-ground.

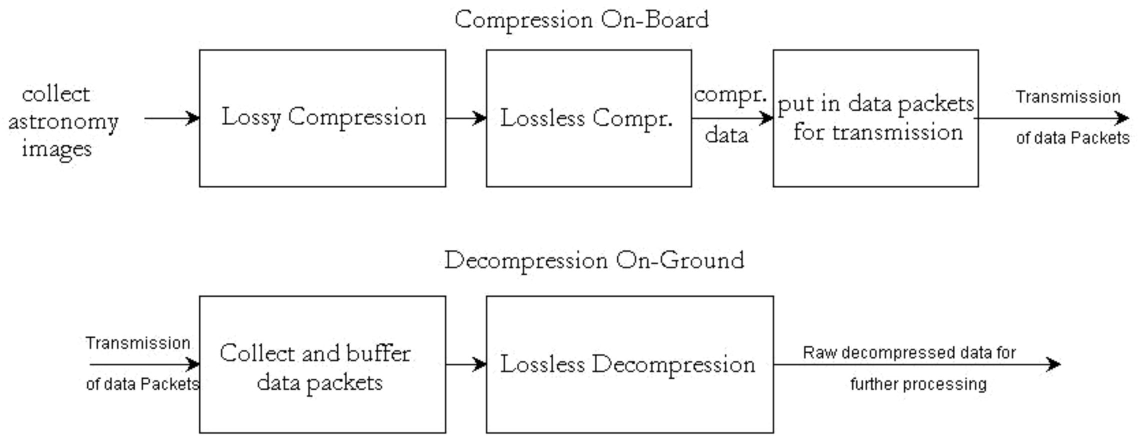
Compression On-Board
collect astronomy images
Lossy Compression
Lossless Compr.
compr. data
put in data packets for transmission
Transmission of data Packets
Decompression On-Ground
Transmission of data Packets
Collect and buffer data packets
Lossless Decompression
Raw decompressed data for further processing

Figure 3: Block diagram of the basic concept

The cameras of the telescope collect images of astronomy objects. They will either be spectroscopy or photometry type images, depending on the type of camera they are from. The data of the images get compressed with lossy and lossless data compression methods. Those methods are specifically designed and improved to get the best compression results. The compressed image data together with telescope control and some additional information is put into a block of data. These blocks are called **C**ompressed **E**ntities (CE). For use of a satellite-to-ground transmission protocol, the obtained compressed entity is then split into several packets, which allow appropriate transmission to earth. This is shown in figure 2 at the upper part.

On-ground the image data has to be reconstructed. The task of reconstructing the images, mainly consist of two parts. The first is to reconstruct the raw data. This is shown with the lower part of figure 2 and that is exactly the task of the software tool, this report is about. Further parts of the on-ground SW take care of analyzing the raw image, control and additional data and some other tasks. Those parts will not be discussed in this report.
The data packets generated on-board the telescope are transmitted to earth. The collection of the packets in first place is done by the on-ground receiving station by hardware which provides the packets in raw data files in an unordered way. The Software, which we implemented - we call it **O**n-**G**round **D**ata **P**rocessing **S**oftware (OGDPSW), then has to collect, check and buffer this data fragments to regain the compressed entities. Once all the data from a compressed entity is received, the CE gets decompressed by running the appropriate decompression algorithms, which are the exact reversals of the lossless compression algorithms used on-board. The use of the compression algorithms is controlled by the operator of the spacecraft and users are informed with release notes to know the applied compression algorithms. The decompressed data is outputted into files on hard disk. Information according to errors during computation is collected and written into logfiles. After the computation the output data can be tested with reference data, if available. In the next chapter we describe how this SW is designed and realized.

4 Software Implementation

This part of the report is about the development of the program itself. We describe the design principles and give some relevant details from the implementation. A class digram and a flowchart of an important part of the software are included in this section. We as well give a short explanation of the tasks of each class implemented. The source code is also well documented with detailed in-source documentation.

4.1 Design of the Program

This software tool can basically be used for handling and decompressing any compressed data. (With modifications if the system is not using the same protocols as the astronomy application we are talking about here).
The program has been implemented in JAVA in an object-oriented way. The benefit of using this language is mainly that the software is immediately available for any given platform. Furthermore, because JAVA is object-oriented, it provides greater reusability and modularity which was especially important for meeting the requirement of flexibility concerning the compression algorithms. This means that simple and quick change or upgrade of compression algorithms must be possible. That was reached by implementing the program with correctly designed and well documented interfaces. Thus the decompression sequence can easily be selected via a regular text file which constitutes the configuration for the decompression. Because of strict time requirements the program was improved for running time. This on the other hand has sometimes led to a trade-off between more efficiency and less beauty in programming. This means for instance that we used arrays instead of vectors even though vector objects would have been more adequate.

A detailed breakdown of the task of the program is the following:

- Read raw data from input files: The packets received by the on-ground receiving station are stored in files. This data has to be read and the packets contained in the data are collected.
- Check files for missing packets: The files are supplied in an unordered way. The files are put into right order and checked for missing data files.
- Check data for transmission errors: Every input file is checked for transmission errors by means of computing a checksum value.
- Appropriate buffering of the data depending on camera source: The checked packet data blocks are temporarily stored according to the data type.
- Reconstruction of the compressed entities: The original compressed data block generated on-board the telescope is reconstructed with the packet data.
- Evaluate decompression sequence from a config-file: The decompression sequence, which is the reversal of the sequence of compression algorithms used on-board, is read from a config file.
- Instantiate decompression-algorithm objects: Creation of a vector of image-decompression objects according to the sequence read from the config file.
- Decompression of the data with these algorithms: The decompression methods contained in the decompression objects is applied to the compressed entity data.
- Output of decompressed data onto files on hard disk: The decompressed raw data blocks are stored in files on hard disk. This data files are now ready for further analysis and processing.
- Appropriate handling of the data in case of errors: The program is capable of handling the data in case of transmission errors, missing and corrupted data.
- Output of a logfile according to the decompression run: Information according to the computation is written to a logfile, which is stored in the output directory.
- Ability to test the result-data of the decompression: The program is able to test the output data with reference data if available.

The program has basically two different running modes: First, a so-called single mode where all the already received packets of data are read and decompressed. Second, a so-called continuous mode where the data is processed consecutively, which means that the program keeps checking for new incoming data. Therefore the input and buffer part of the program has been realized with threads. The threads either run just once or consecutively depending on the modus the program gets started.

4.1.1 Input and Collection of data packets

The receiving station will put the received data packets into files, which then are present in an unordered way in the working directory on a hard disk. The program starts with reading the file list and the files itself. After reading the raw data from the files a CRC-16 CCITT checksum value is computed. This value is compared with the value which was generated and added to the packets on-board. If the checksum value differs from the value computed on-board the corresponding decompressed data is marked with *ERROR* and a log file is created at the end of the computation.
If the program notices missing files, which means packets are missing, it will determine the corresponding compressed entities. These are not processed but saved in one single file

instead. That file is marked with *INCOMPLETE* and information about number, size and input files is stored in the error log file.

If the checksum is correct the packet header is evaluated.

This header mainly stores the following information:

- The total number of packets the CE has been split to
- The number (index) of this single packet
- Whether the CE was processed through **D**igital **S**ignal **P**rocessor (DSP) 1 or 2

A new CE is created every time a packet with index 1 is received. The CE is assigned to a CE information object which stores the unique internal CE number, the DSP, block and index information. At any time there will be 2 CE information objects - one for each DSP. Those are need for unique internal identification. All other packets are checked for the information in the packet header and assigned to the right CE. If missing files are detected, and it is not clear where a packet has to be assigned to, a new CE object is created as well. That will be stored in the error logfile. Once a CE object is completely set up with packets, the CE-header is extracted. Accurate information how the CE-header is set up can be found in [3]. Then the data without packet and CE header information gets decompressed.

4.1.2 Decompression and Output of data

As soon as the data in a CE has been put together completely the decompression of the CE will start. Several different compression algorithms are used on-board to compress the data. The system uses lossy and lossless compression algorithms. The exact reversals of the lossless compression algorithms are applied to the compressed data on-ground to get the original data, after lossy compression, back. A description of the basic function of the compression algorithms implemented so far can be found in the appendix. There is lots of literature where these algorithms are described closely [2,11,13]. New compression algorithms can easily be added to the program by adding a class with the new algorithm which implements the IImgDecompression interface. The selection of the used compression algorithms for a given set of data can be done with a config-file or even dynamically during the data reconstruction sequence.
The data in a compressed entity consists of two parts: the **C**ompressed **D**EC/MEC **H**eader (CDH) and the **C**ompressed **S**cience **D**ata (CSD), which itself contains the **C**ompressed **S**cience (CS) the **R**aw **C**hannel (RC) data. The CDH is the final Header information, which contains control information. The exact form of this header and all its fields and their meaning is described precisely in [3]. The CS contains the real image data. The RC is some additional raw channel information. For each type of camera, that means spectroscopy or photometry, different sets of compression algorithms on-board and thus different sets of decompression algorithms on-ground are used.
If there are options or parameters used for decompression algorithms they are added either at the beginning or at the end of the data. Some of the parameters used are not contained in the data and thus come from outside. Therefore the IImgDecompression interface constitutes a parameter array which is assigned to the decompression algorithm.
When all the algorithms of a decompression sequence finish, the decompressed data is saved in files arranged in subdirectories in the output directory on hard disk. Each subdirectory is for one CE. The output directory has date and time information to distinguish it from others, if more decompression runs are done. The files have the file extension *.raw*. That means

CDH.raw, CS.raw, and RC.raw. If upon program start the –f option is chosen the CS data is saved in frames rather than in one single file. The frame size depends on what camera the CE is from. The filenames in that case are Frame001.raw, Frame002.raw, etc.
Any error happening during the computation will be saved in the error log file which is then placed into the output directory on the hard disk. In such a case also the decompressed data is accordingly marked. If, due to an error, the data cannot be processed at all it is saved in raw compressed form and the user is warned.

4.2 Description of classes and class diagram

In this section we will give a short description of the implemented classes. Figure 4 and 5 show the class diagrams to give an overview. Note that these diagrams are not intended to be detailed class diagrams, but to show every class implemented and the most important fields and operations of the classes.
The program is separated into two packages, **dataprocessing.decompression** which includes all classes for the compression algorithms and **dataprocessing** which includes all control and entity classes.

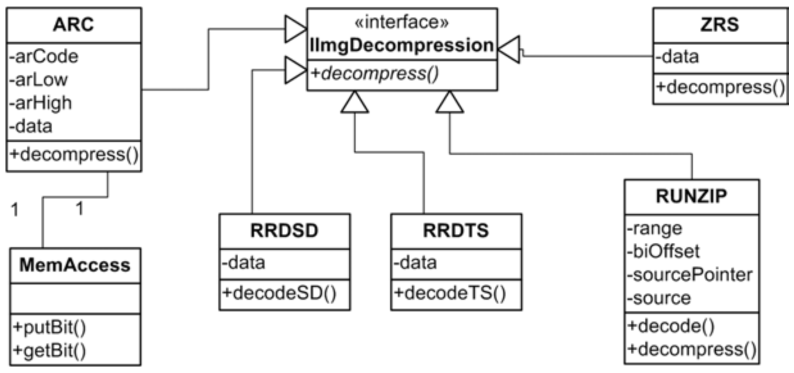

Figure 4: Class diagram of dataprocessing.decompression package

All Decompression algorithms are associated to the dataprocessing.decompression package. Every class that implements a new algorithm needs to implement the IImgDecompression Interface. If done so, simple change and upgrade of Compression methods are guaranteed. In the Config-File the algorithms get chosen by the class name.

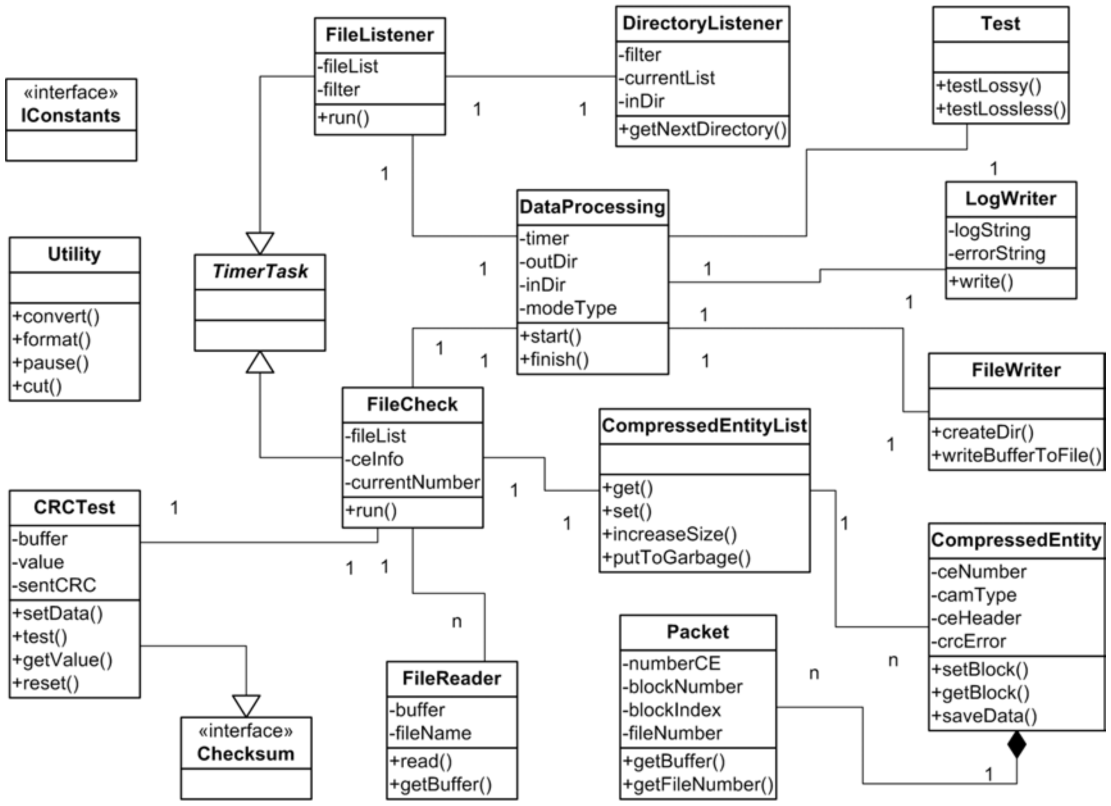

Figure 5: Class diagram of dataprocessing package

- **DataProcessing** is the main control class of the program. This class has the main function so everything gets started from there. Upon program start, input and output directories are evaluated as well as OS-depending information. The decompression sequence from the config-file is interpreted with this class as well. Before terminating the program, the finish function is invoked which ensures that all data is processed correctly and none gets lost and also the log file is written with a **LogWriter** object if this is necessary. If upon program start one of the test options is selected a new **Test** object is created after the reconstruction of the data.
- **LogWriter** writes information according to the decompression run to a log file.
- **Test** allows for testing the resulting data of the decompression run. A Test object checks the reconstructed data with a reference data and computes reconstruction error measures.
- **FileListener** checks the working directory for incoming data. The files get selected via a **FileFilter** object which is initialized with the base filename given as a parameter at program start.
- **DirectoryListener** is used by FileListener, to read the directory information and provide the current reading directory which is especially important in case of –d mode.
- **FileReader** class is for reading the input files and for evaluating the checksum values.
- **CRCTest** does the computation of the CRC16 value.

- **FileCheck** is the class which controls all the handling of the packets and appropriate buffering of the input data. A unique CE-number is created and assigned to the data packets. FileCheck evaluates the packet headers and assigns the **Packet** objects to the adequate **CompressedEntity** objects which themselves are stored in the **CompressedEntityList.** The end of this chapter shows Figure 6 which represents an overview flow chart of the collecting and buffering sequence.
- **CompressedEntityList** is the data structure for saving the CE's.
- **CompressedEntity** is the class for combining all the data and operations for a CE. As soon as all packets are set in a CompressedEntity object, it starts its own decompression through invoking the decompression method in the **IImgDecompression** objects selected during evaluation of the decompression sequence stated in the config-file. This sequence, finally saved in an array of IImgDecompression vectors, starts the right decompression algorithms by using the classes in **dataprocessing.decompression.**
- **IImgDecompression** interface is necessary for a standardized use of compression algorithms.
- **FileWriter** saves the data onto hard disk upon decompression.
- **Utility** combines useful methods such as methods for transforming between little and big endian and so on.
- **IConstants** interface holds all constants, being relevant for the program, in one place**.**

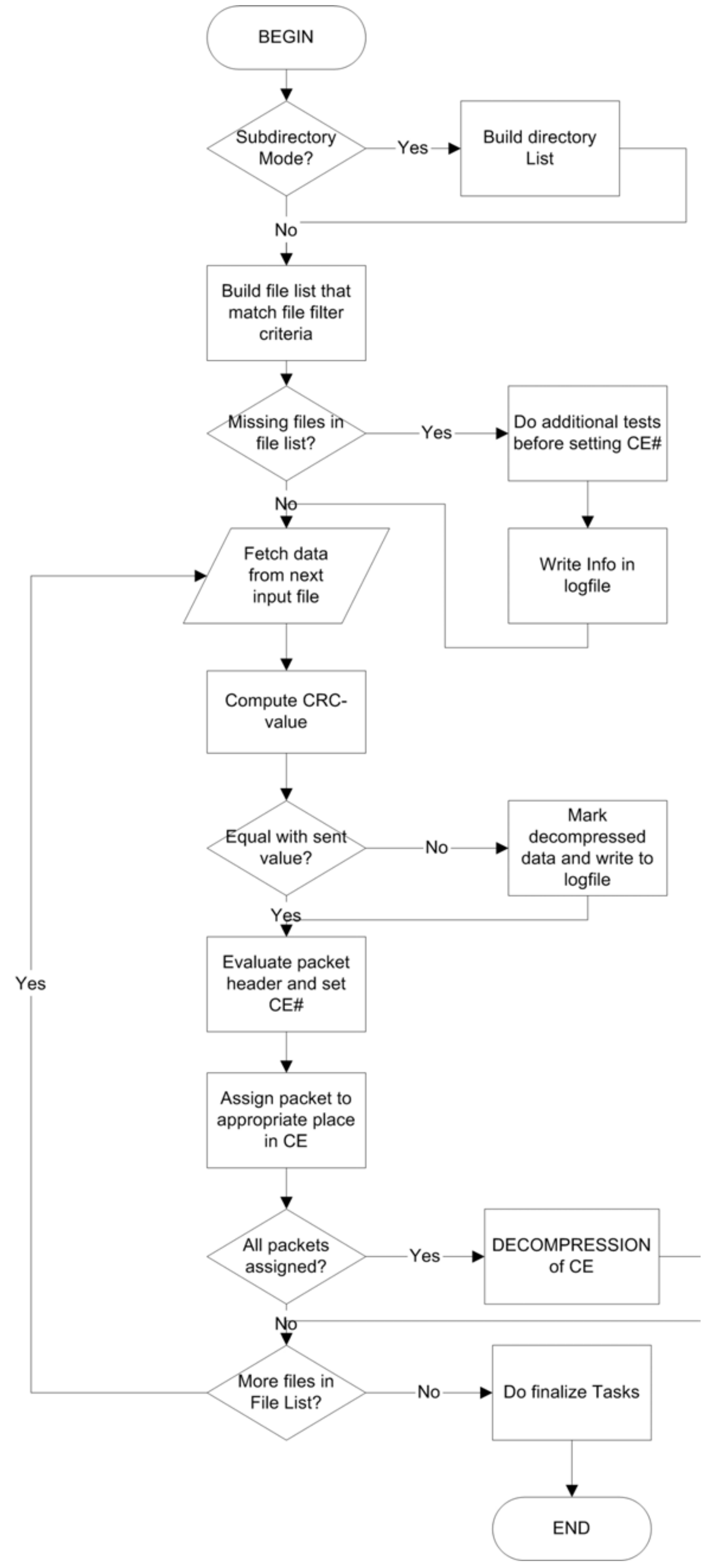
BEGIN
Subdirectory Mode?
Yes
Build directory List
No
Build file list that match file filter criteria
Missing files in file list?
Yes
Do additional tests before setting CE#
Write Info in logfile
No
Fetch data from next input file
Compute CRC-value
Equal with sent value?
No
Mark decompressed data and write to logfile
Yes
Evaluate packet header and set CE#
Assign packet to appropriate place in CE
All packets assigned?
Yes
DECOMPRESSION of CE
No
More files in File List?
Yes
No
Do finalize Tasks
END

Figure 6: Flowchart of data buffering

5 Evaluation of the Software

Several tests have been done with the software. For this matter test data files were created with the telescope simulator program. This program is exactly the same software later running on the DSP of the telescope. On-ground it only differs in the fact, that it creates random image data instead of handling real image data. The process of lossy and lossless compression and the alignment of the data in packets is the same as it will be at the spacecraft later on. This simulator program outputs two types of reference data. First of all it creates the generate-files which are the frames of the random images before lossy compression. It also outputs the three main data files (DEC/MEC Header, Image data, Raw Channel data) after lossy compression but before lossless compression. Hence it's possible to compare the decompressed files generated by the on-ground software with the reference data from the telescope simulator program. Generally, the test was to process the images from 24 hours, which is around 160 MB of CE-data, without any failure or errors for the decompression. All possible running modes of the program have been used in this test, to check them independently.
During this correctness test, the program was also tested for time performance. This was done on two different machines, one Linux and one Windows platform both on the same hardware, which was basically the following machine:
AMD XP2600+ processor (1916 MHz) running on an ASUS-A7N8X main board with 512MB DDRAM.
The time performance on Linux is better than on Windows, especially on higher data rate. This is due to the file management of Windows. Thus more time for reading input data is needed on the Windows machine. The time for reading input files is about 30% of the total computation time when processing 160MB of CE data in Windows. Usually the time rate between reading and decompression is about 3:97. We recommend, saving the files in subdirectories and use the –d option for the program to improve time performance, particularly when using Windows. The tests have shown that it is the best way to save the files in directories which hold about 2500 files. It is to mention that the time needed for processing the data will slightly depend on the shape of the data. Generally the time needed for the decompression of 160MB of data, which later in reality would be approximately the amount of data from images collected in 24 hours, is around 90 minutes. The exact values are shown in Figure 7 and 8.

# CE	# Files	Data size [MB]	Time perfomance in Windows [sec]	Time perfomance in Linux [sec]	Time perf. in Win. with subdirectories [sec]
98718	200164	154	9150	5760	6145
49389	100082	77	4963	2923	3642
24695	50041	38	2431	1450	1312
12347	24560	19	889	723	452
6059	12280	9,5	270	272	272
3029	6140	4,7	153	122	150
302	614	0,5	14	12	14

Figure 7: Time performance table

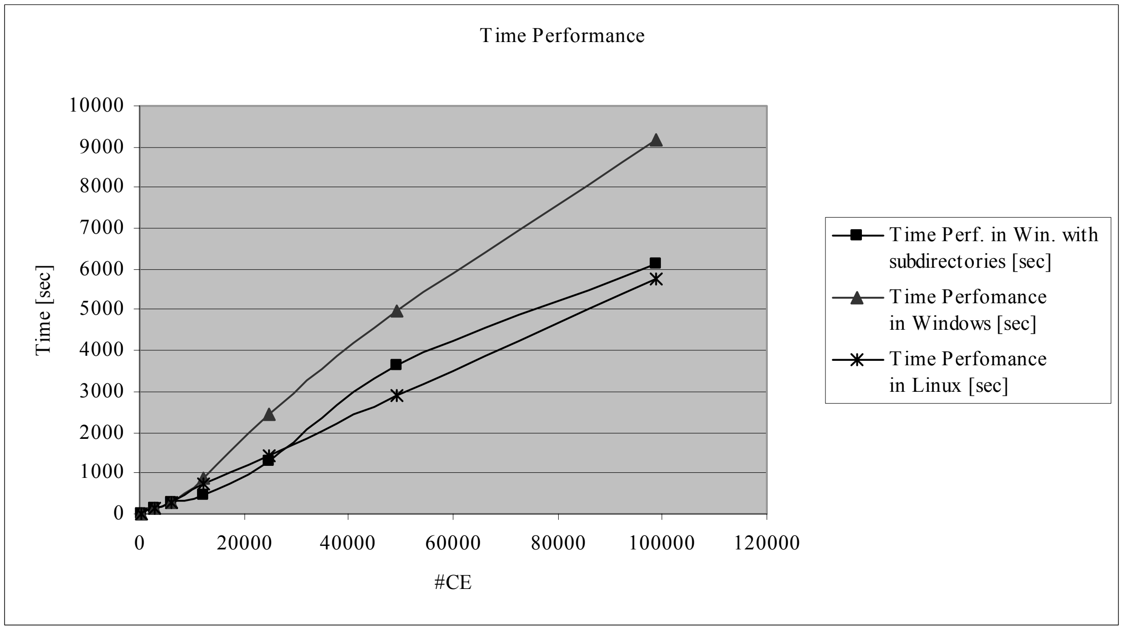

Figure 8: Time performance diagram

The program was also tested on how to deal with errors in the input data. This could be missing data packets or corrupted data. Generally, in those cases the data will either be decompressed and marked or saved unprocessed. An error logfile is created with accurate information about the problem.

If the program is started in –t testmode the software computes a reconstruction error measure between the reference files created from the telescope simulator and the decompressed data. The type of values computed are the ones described in section 2.2. These measurements describe the amount of information lost in a compression/decompression run. This is especially useful to apply on a lossy compression run to evaluate the lossy compression method. For instance one can evaluate the compression method with the reconstruction error versus the achieved compression ratio. Therefore we compute the reconstruction error between the original data files stored in the generate-files and the files before lossless compression or the files after decompression, assuming no fault during data processing with the OGDPSW. An example of images before and after lossy compression is shown in Figure 9 - 11. Figure 12 shows the computed reconstruction error measures. Please note that these infrared type images are post processed with virtual colours. The lossy compression method in this case was to compute the average frame out of four frames.

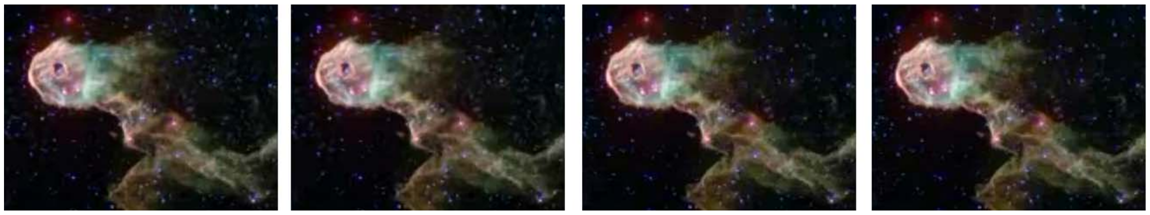

Figure 9: Original image sequence of an astronomy object 1

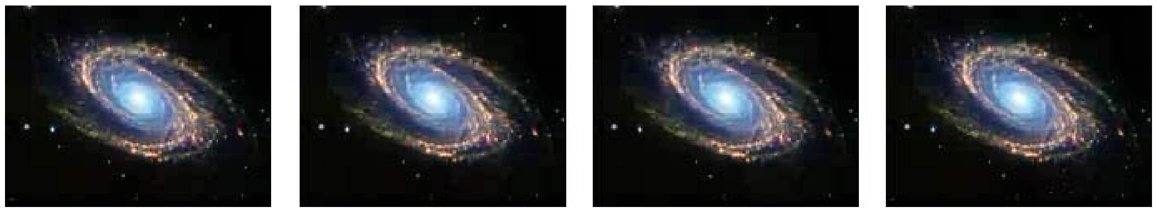

Figure 10: Original image sequence of an astronomy object 2

Figure 11: Image 1 and 2 after lossy compression

	MAE	RMSE	SNR	PSNR
Max values in Image Sequence 1	7,6	12,0	19,3	26,6
Max values in Image Sequence 2	4,9	8,6	22,0	29,4

Figure 12: Reconstruction error measures for both sequences

6 Conclusion and Outlook

This paper describes the design of a software tool which can be used for handling and decompressing data which has been compressed, using various compression algorithms, and arranged in a way, which allows for transmission over channels like the internet or a satellite link for instance. It is described for use in an astronomy application. Hereby is it a part of a larger software package for handling astronomy information. This part of the Software provides the functionality for the first processing step of the received data from the telescope. This SW does all the necessary steps to provide raw data which later can be semantically analysed and further processed. It evaluates all parameter information for the appropriate decompression of the data. It supplies the on-ground system not just with the image data but also with additional and control information gathered at the sender.
Further parts of the on-ground software are using this raw data and deal with reconstructing and analysing the real images collected with the cameras and sensors on-board the telescope.

Acknowledgements

This work was done at the Vienna University of Technology, Institute of Computer-Aided Automation, **P**attern **R**ecognition and **I**mage **P**rocessing Group (PRIP) in cooperation with the Institute for Astronomy of the University of Vienna. I wish to thank my advisors Dr. Robert Sablatnig and DI. Ahmed Nabil Belbachir for their support.

References

[1] Official ESA Website at: http://www.esa.int/science/herschel

[2] G. Held, "Data and image compression: Tools and techniques", J. Wiley and Sons, England 1996

[3] A.N. Belbachir, "SPU HIGH LEVEL SOFTWARE Specification Document'', Document Ref.: PACS-TW-GS-001 Issue: 4.1, Austria 2003

[4] J.Sanchez and M.P. Canton, "Space Image Processing", CRC Press, Florida 1999

[5] Data compression Website at: http://www.datacompression.info

[6] V. Bhaskaran and K. Konstantinides, "Image and Video Compression standards", Kluwer, Boston 1997

[7] J. A. Storer, "Image and Text Compression", Kluwer International Series, Boston 1992

[8] D.A. Lyon, "Image Processing in Java", Prentice-Hall 1999

[9] A. Bovik, "Handbook of Image and Video Processing", Academic Press 2000

[10] A.N. Belbachir and H. Bischof, "On Board data compression", PRIP Technical Report Austria 2003

[11] K. Sayood, “Introduction to data compression”, Academic Press 1996

[12] Lossless Data Compression, CCSDS 121.0-B-1, Blue Book, Issue 1, May 1997

[13] J.A. Storer, “Data compression: methods and theory”, Computer Science Press 1988

[14] M. Nelson and J.I. Gailly, “The data compression book”, M&T Books New York, NY 1995

[15] D. Salomon, “Data Compression: The Complete Reference”, Springer 2000

[16] A. Moffat and A. Turpin, “Compression and Coding Algorithms”, Kluwer Academic Publishers 2002

[17] I.S. Glass, “Handbook of Infrared Astronomy”, Cambridge University Press, Oct.1999

[18] W. Wijmans and P. Armbruster, “Data Compression techniques for Space Applications”, DASIA’96, Italy 1996

[19] M. Datcu and G. Schwarz, “Advanced Image Compression: Specific Topics for Space Applications”, DSP’98, Int. Workshop on DSP techniques for Space Applications 1998

[20] J. Blommaert et al, “CAM - The ISO Camera”, ISO Handbook Volume III, V2, June 2003

[21] http://www.iso.vilspa.esa.es

[22] http://irsa.ipac.caltech.edu/IRASdocs/iras.html

[23] http://www.spitzer.caltech.edu/

[24] A. Poglitsch et al, “The Photoconductor Array Camera & Spectrometer (PACS) for the Far Infrared and Submillimetre Telescope (FIRST)”, Germany 2000

A Basics of Compression Algorithms used

In this chapter we want to explain the basic functioning of the already implemented compression algorithms.

A.1 Arithmetic Coding

Arithmetic coding is a compression technique where each symbol in a set of symbols is assigned to an interval between 0 and 1 according to its probability of occurrence in the message. Arithmetic coding encodes a stream of data into a large binary fraction. It can achieve near-optimal entropy encoding [15,16].

The process of encoding can basically be described as follows:

1. Determine the probability of each symbol in the message, which means determine the frequency of the symbol.
2. Build a table which lists all the symbols and their assigned probabilities. That is build a range for every symbol which is sum of occurrence of the symbol divided by the total number of all symbols. The sum of all ranges is always 1. This table must be used in the same order for encoding as well as decoding.
3. Take the first symbol in the message and select the range associated with it.
4. Select the next symbol and multiply the range from the last symbol with the low and the high bounds of the range of the current symbol. This results in a new range. Add the results to the low value of the prior symbol.
5. Continue doing this until all symbols are processed.
6. Select any number of the resulting range. This is the encoded value.

The computation of the coded value can be summarized as follows:

- Start with range $[0.0 - 1.0[$
- Then for $i > 1$ where i is the index of the symbols in the message.

$$low[i] = low[i-1] + range[i-1] * low_{assocRange}[i] \qquad \text{where:} \quad range[i] = high[i] - low[i]$$
$$high[i] = low[i-1] + range[i-1] * high_{assocRange}[i]$$

The following example shows an encoding procedure:

Symbol	Assoc. Range		Input	Current Interval
a	[0.0 - 0.4[			[0.0 - 1.0[
b	[0.4 - 0.6[		b	[0.4 - 0.6[
c	[0.6 – 0.9[		c	[0.52 – 0.58[
EOF	[0.9 – 1.0[		a	[0.52 - 0.544[
			EOF	[0.5416 - 0.5440[

Figure 12: Example for arithmetic coding with result values

The coded value representing this short message is any value in the range 0.5416 to 0.5440. Let’s say for example 0.542.

The process of decoding which uses the same formulas can be written as follows:

1. Check the range the coded message falls in. In our case 0.542 is inside 0.4 – 0.6 range, which yields the first symbol b.
2. To get the next symbol we have to subtract the low value of the first symbol. This brings us to 0.142. Because the range of b was 0.2 we have to divide the new value by 0.2 which gets us to a value of 0.7. That value falls into the range 0.6 – 0.9 resulting in the second symbol c.
3. We have to repeat this until we find EOF. This leads us to 0.367 which is in range 0.0 – 0.4 resulting in symbol a. 0.367 divided by the range of symbol a, which is 0.4 results in 0.9175 which is in range of the symbol EOF.

As the length of the message increases the computed range gets narrower. It is obvious that, this would soon exceed the precision available at the computer.
To perform arithmetic coding on a computer we have to use finite-precision binary integer arithmetic. More about arithmetic coding and how to implement it can be found in [2,3,11].

A.2 RZIP Compression

RZIP is a compression algorithm specially designed to achieve maximum compression ratios for little processing power. It is a lot faster and much more efficient than encoders using DPCM [3,12]. Hence it’s optimally feasible for this task.

The Encoding in RZIP goes as follows:

Hereby is:	SOURCE:	source data of size SSIZE
	ALPHA:	temporary working buffer of size SSIZE which is filled with 0 integers initially
	DEST:	destination buffer where the coded data is written to
	RANGEWIDTH:	the # of bits, the offset of a symbol is encoded in
	RANGE:	the range to search for one symbol – which sets the offset
	YES/NO:	1 bit symbol to determine if new symbol is started or not

Symbols in destination are 32 bits, offset is RANGEWIDTH bits, YES/NO is 1 bit wide

1. Select first symbol and output it immediately. Set ALPHA buffer to 1 at this index.
2. Iterate through source and check if same symbol can be found somewhere again within RANGE. If yes code YES and code an offset, which represents the number of index between last and current symbol of this kind. Increase offset just if index has a 1 in the ALPHA buffer.
3. If no symbol of the same kind can be found within RANGE code NO.
4. Select next symbol in source where ALPHA is 0.
5. Goto 2 until source buffer is finished.

For example this source data with its ALPHA buffer:

Source data	A	B	B	C	A	B	A	D
Alpha array before coding A	0	0	0	0	0	0	0	0
Alpha array after coding A	1	0	0	0	1	0	1	0

Destination data after coding symbol A:	AY3Y1N
At the end the coded data will look like this:	AY3Y1NBY0Y1NCNDN

More information on the method of RZIP compression can be found in [3].

A.3 Simple Zero Repetition Suppression (ZRS)

ZRS is a compression algorithm to get rid of zeroes in the data. All it does is to ignore zero symbols in the buffer and instead output the number of zero symbols between two nonzero symbols. Note that this algorithm is only useful if there are more zero symbols than nonzero symbols in the data [3,14].

How this procedure works can easily be shown in an example:

A buffer like this	A 0 B 0 0 0 0 C 0 0 D
Is coded to this:	A 0 B 1 C 4 D 2

Zero count for the first symbol is always zero. Offset and symbols in destination are written in the same quantity of bits.

A.4 Redundancy Reduction

Before any of the compression algorithms are used two different methods for redundancy reduction are applied to the data. These two methods are **S**tatic+**D**ynamic and **T**emporal+**S**patial **R**edundancy **R**eduction (SDRRED, TSRRED). These algorithms are described exactly in [3].

B Abbreviations

ARC	Arithmetic coding
CDH	Compressed DEC/MEC Header
CE	Compressed entity
CSD	Compressed science data
CR	Compression ratio
CDH	Compressed DEC/MEC header
DSP1	Digital signal processor 1
DSP2	Digital signal processor 2
DPCM	Differential pulse code modulation
HSO	Herschel space observatory
ISO	Infrared Space Observatory
IR	Infrared
IRAS	InfraRed Astronomical Satellite
MAE	Mean absolute error
OGDPSW	On-ground data processing software
PSNR	Peak signal to noise ratio
PACS	Photodetector array camera and spectrometer
RMSE	Root mean square error
RC	Raw channel data
SNR	Signal to noise ratio
SW	Software
SDRRED	Static and dynamic redundancy reduction
TSRRED	Temporal and spatial redundancy reduction
ZRS	Zero repetition suppression

C Manual of the Program

The name of the program (JAVA archive) is DataProcessing.jar. If one types
java –jar DataProcessing.jar –help the following help screen appears:

```
Usage: java -jar DataProcessing.jar MODE [OPTION] FILEBASE
MODE      -o for decompression of fixed set of files
          -c for continuous decompression (keep checking for new files
             and stop after 3 minutes without new data!)

OPTION    -f for framebased Decompression of science data
          -d for including files in subdirectories of working directory
          -t for doing a testrun after decompression
          -l for doing a testrun which computes also reconstruction error
             measures for lossy compression

FILEBASE  basename of files to decompress


This help screen shows all the possible running modes with all the options
available and describes them briefly.
```