University of South Australia

Institute for
Telecommunications
Research

Processing MODIS Data for Fire Detection in Australia

Kendy Kutzner

Supervisors:
Prof. William Cowley, ITR
Michael Sloman, ITR

School of Electrical and
Information Engineering

December 7, 2001

I declare that this work does not incorporate any material previously published or written by another author except where due reference is made in the text.

[illegible]

Adelaide, December 7, 2001 Kendy Kutzner

This work is submitted to the Institute for Telecommunications Research, Australia, in order to fulfill the requirements for

- Studienarbeit to reach the degree of 'Diplomingenieur Elektrotechnik' at the Chemnitz University of Technology, Germany
- Final year project (hons) at the school of Electrical and Information Engineering, University of South Australia

The honors part of the work is especially present in section 4.

Contents

List of Figures

List of Tables

1 Abstract

The aim of this work was to use remote sensing data from the MODIS instrument of the Terra satellite to detect bush fires in Australia. This included preprocessing the demodulator output, bit synchronization and reassembly of data packets. IMAPP was used to do the geolocation and data calibration. The fire detection used a combination of fixed threshold techniques with difference tests and background comparisons. The results were projected in a rectangular latitude/longitude map to remedy the bow tie effect. Algorithms were implemented in C and Matlab. It proved to be possible to detect fires in the available data. The results were compared with fire detections done by NASA and fire detections based on other sensors and found to be very similar.

2 Introduction

This section should give an overview over problems this work is dealing with. It begins with a general introduction why remote sensing is useful for firedetection. Then the used conventions and file formats are explained and at the end some of the problems are described in detail.

2.1 South Australia and Bush Fires

South Australia is the third largest state in Australia with an area of more than $9.84 \cdot 10^{11} m^2$ [7]. It is often referred as the driest state in the driest continent. Its large agricultural, shrub and forested areas posing a high fire hazard in most summers. There were 3204 hectares burnt and over 5 million Australian Dollars lost in rural fires in the 1999/2000 summer alone[3].

This should make clear the need for fast fire detection. Fire reporting is difficult because of the vast area and sparse population in most of the state. This work tries to determine the possibility of using remote sensing data to support the early fire detection.

2.2 ITR and ASTRA

The Institute for Communications Research (ITR) has got the ASTRA since December 1984[11]. ASTRA stands for Automatic Satellite Tracking Research Antenna and is originally from NASA. It is a 6.8m hydraulical steerable parabolic dish and is currently configured for X band. It is possible to receive Earth Resource Satellites such as ERS, JERS, SPOT, MOS, LANDSAT and RADARSAT[11]. In this work the interest lies on the satellites Terra (formerly EOS-AM) and Aqua (EOS-PM).

After reception with ASTRA the signal gets immediately demodulated with ERSDEM2, a configurable demodulator capable of data streams up to 150 megabit per second developed at ITR.

2.3 Terra and MODIS

The Terra satellite is part of NASA's Earth Observing System (EOS). It was launched in December 1999 and is referred as the flagship of the EOS. Since February 2000 it is fully operational. On of the main instrument it carries is the Moderate Resolution Imaging Spectroradiometer or MODIS. This sensor sees the entire earth surface every two days and acquires data in 36 different bands.

MODIS is meant to be of use for a very broad range of scientists. Many of its channels are especially designed for terrestrial, atmospheric or ocean phenomenology. The EOS-PM or Aqua platform will carry the same instrument. Aqua will be launched, according to [23], in March 2002. Then the already good coverage will be even better. Until now, every day one overpass can be recorded in the morning, with Aqua there will be one in the afternoon (That's why the satellite was called -PM).

Data collected by MODIS is transmitted immediatly via the direct broadcast facility or can be stored and transfered to the earth later in direct playback mode. ITR has the capability to receive the continuous direct broadcast datastream from the MODIS instrument.

This work is the trial to process the data above the level 1b stage and maybe provide useful results to organizations like the Country Fire Service.

The goal was to automatically detect bush fires from received data.

2.4 Conventions and Abbreviations

For a list of abbreviations please refer to appendix B.

'Command Line' usually refers to a `sh` shell on a Sun OS or Linux computer unless otherwise stated.

The most often used symbols include:

λ	Center wavelength of a channel (in μm)
ρ_λ	Reflectance at wavelength λ
B_λ	Radiance at wavelength λ
T_λ	Brightness temperature converted from B_λ
ϕ	Latitude values
θ	Longitude values

2.5 Overview over Data flow

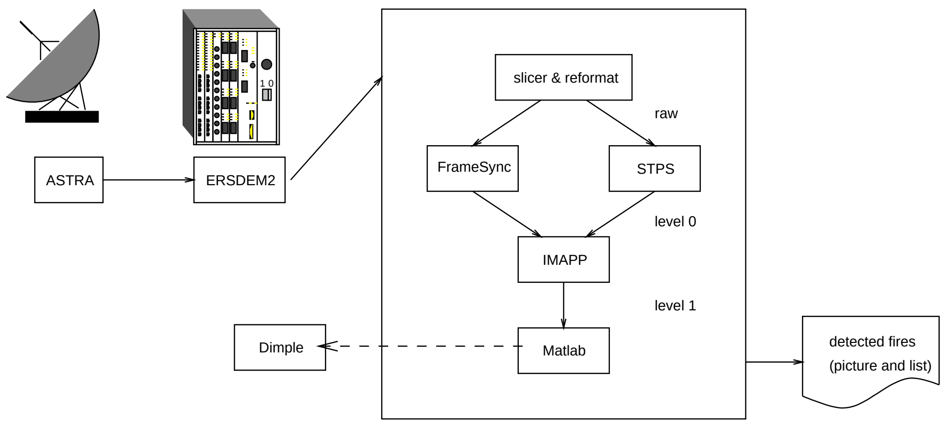
Firedetection does not work immediately with the received data. The files have to be processed a few steps. To make these processing stages independent of each other different levels have been defined and widely accepted.

Raw data is just recorded from the satellite and not processed in any way. *Level 0* data is defined as the reconstructed CCSDS packets (see section 3.2.2) with all communication artifacts including duplicate and erroneous packets removed. *Level 1a* is an intermediate product based on HDF (see section 2.6.3) produced by IMAPP (see section 3.5). *Level 1b* data is the radiometrically calibrated data. Furthermore geolocation information is available in the level 1b product. This includes ancillary data like the position of the spacecraft or sensor and solar angles.

In level 1b products enough information is available to do a successful fire detection.

The flow of data can be seen in figure 1. Raw data is received by ASTRA and demodulated with ERSDEM2. These steps are not part of this report. This work starts at the preprocessing stages when raw files are touched for the first time in the conversion to level 0 data. Preprocessing includes the exclusion of obviously bad data and the bit reversion. Level 0 data is produced by FrameSync or STPS (see sections 3.3 and 3.4). IMAPP is responsible for steps to level 1b. The actual fire detection is based on this output and is done in Matlab.

Figure 1: Flow of data from ASTRA to fire detection result



2.6 Used File Formats

2.6.1 Raw recorded data

After reception of the data by ASTRA and demodulation by ERSDEM2 data is recorded to computer files. These files have no particular format and contain a concatenation of bits. Usually these files carry the extension `raw`. Due to the Terra data rate of 13.125 megabit per second in the used direct broadcast mode and a average overpass duration like 12 minutes the typical file size is around one gigabyte. A more detailed description of the data and problems associated with it are in section 3.1.

2.6.2 Production Data Sets

Data in level 0 is often called production data sets. These data sets consists of consecutive CCSDS packets (see section 3.2.2). These packet stream should not contain any artifacts from the communication any more. All erroneous and doubled packets should be removed.

Some programs like further processing software from NASA requires additional headers containing meta information (EDOS headers), but in this project IMAPP (see section 3.5) is used for level 1 processing and doesn't need such headers.

Since these files don't contain the protocol overhead they are a little bit smaller than raw data files (see section 2.6.5).

The extension of these files is usually `.pds` for production data set or `.ccsds` because they contain CCSDS packets.

2.6.3 Hierarchical Data Format

The Hierarchical Data Format (HDF) was developed by the National Center for Supercomputing Applications. It is a open file format, it is well documented and supported and mainly used for scientific data exchange and archival.

There are two entirely different and incompatible formats. The newer one is HDF version five. In this project however only version four is used, it is still fully supported by NCSA and many available tools operate on this format.

One of the main features of HDF is its support to store meta data or attributes with the data. This is used for example for human readable description of data sets or storage of additional scaling values.

Another interesting feature is the ability to store multidimensional data sets. Here the maximum dimension used is three, since many two dimensional bands are stored together in one data set.

An advantage of HDF over other file formats is the strict need to used the provided interface. All accesses to the data are done through a library and no low level knowledge of the file format is necessary.

2.6.4 Other used File Formats

Parts of this project require the store of image data. For this purpose the format TIFF (Tagged Image File Format) was chosen. This has several reasons including the wide support for this format across different computer platforms and software packages. It is capable of lossless data compression and stores 24 bit color images.

Text files include the output of the fire detection algorithm, the configuration files mentioned and all source files for Matlab and C. Text files are stored in the Unix way for line endings with 0x10 as a line end marker. To convert them to any DOS/Windows based platform the tool `unix2dos <filename>` can be used.

The text file produced by the fire detection script are formated in a way that further processing is easy. For example to get the number of detected fires one can type

```
grep -v ^# filename | wc -l
```

at the command line prompt. The file header contains information about the processing done and the proportions of the produced image. Input and output file names are stated. After the header for each fire pixel found one line is printed. It contains the latitude and longitude values of the pixel as well as the position of the fire pixel in the final map.

2.6.5 File Sizes

Remote sensing data tends to be large, especially with higher resolution. To get an idea of file sizes, see table 1. All programs and algorithms dealing with these files must be aware of this.

Table 1: Examples of File Sizes

recording date	29/11/2001	05/12/2001
initial file size[a]	411,041,792	312,475,648
after FrameSync[b]	330,155,562	250,984,764
level 1a	98,713,604	77,012,996
geolocation data	50,765,783	38,511,453
1000m data	287,492,136	218,165,563
500m data	230,296,084	174,759,533
250m data	239,500,060	181,742,949
resulting .tiff file	8,072,286	3,074,088

[a]raw data
[b]level 0

2.7 Bow Tie Effect and Map Projections

The MODIS instrument has the nominal spatial resolution only near the nadir line. As a result of the motion of the scanning mirror in the instrument, data elements at the edge of the 110 degree swath width have a size of more than double of the nominal one[26]. This leads to overlapping of swaths and to a distorted image at the edges. Figure 2 is an example of the Bow Tie effect. It was recorded on 30/08/2001 and shows the Eerie and York

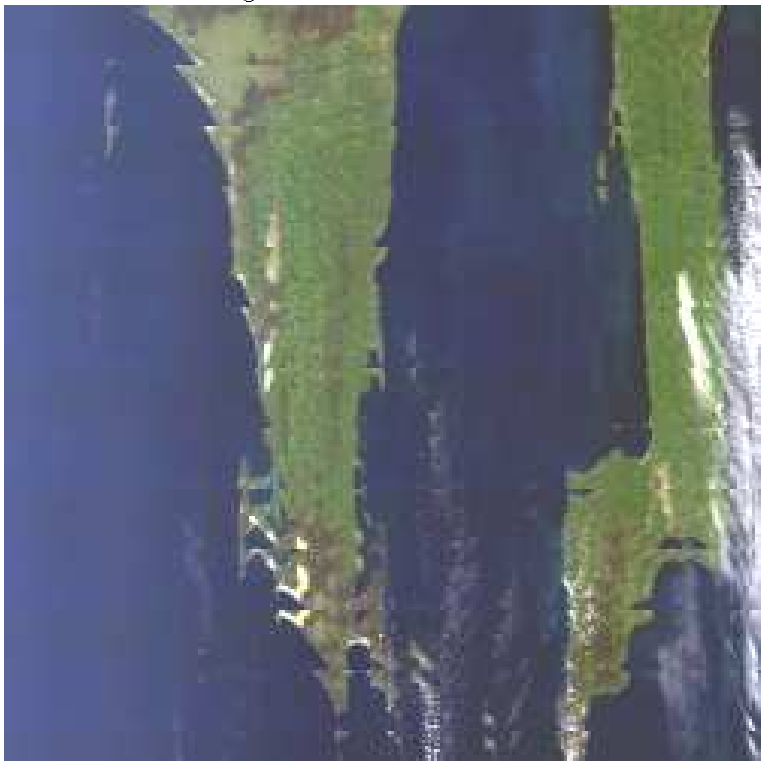
Peninsula in South Australia in the 500m uncalibrated color. Note the wrong aspect ratio as a result of non square picture elements from MODIS. Another unpleasant artifact is that the image orientation is not north but in the flight path of the satellite.

IMAPP provides the geolocation data for every 1km pixel. This information can be used to correct both Bow Tie and distortion effects.

Earth's near spherical surface is not easily projected to a plane. Many different approaches are known. Two common ones are the UTM (Universal Transverse Mercator) and rectangular latitude/longitude. Good explanation can be found in [27]. For conversion see for example [29]. The first provides much better results for smaller areas but proved not practical for recorded images with sizes up to 3420x2332km.

Rectangular latitude/longitude maps the spherical surface to a rectangle with latitude and longitude as X and Y coordinates. This has the apparent disadvantage of distortion near the poles (a point is mapped to a line) but this is not a huge problem in the land area of Australia with latitude values less than 40 degrees.

Figure 2: Bow Tie Effect



3 Data Processing to Level 1b

3.1 Preprocessing

After recording the raw data from the satellites direct broadcast facility a number of processing steps are necessary. The first is the preprocessing to get rid of larger amounts of bad data recorded from the demodulator. There are blocks of unusable data at the beginning and the end of each recording. Further bad data blocks may be there when the tracking of the satellite fails for some reason or when the satellite's transmitter is switched off for the reception of deep space probes.

This is done with the small program `slicer`, which is based on a similar program from Ricky Luppino. Source code can be seen in appendix E.2. It performs a kind of an auto correlation of the raw data. If this correlation peaks at 1024 because of the repeated sync markers (section 3.2.2), the program assumes good data and passes the data to the next steps.

Another preprocessing is necessary for the use of the STPS package described in section 3.4. This program can't deal with byte reversed data available in ITR. The program `reformat` (see section 3.4.1) cancels this reversion.

3.2 Introduction into Frame Synchronization

As described before, the output of previous stages is the concatenation of bits. Until now, no alignment is done and none of the bits has a meaning. In the files the raw recording from all broadcast communication of the satellite is saved. Another name for this type of data is raw telemetry channel data.

The next task is to byte align these bits. After this, the protocols used have to be understood and the data we are interested in can be extracted.

3.2.1 Bit Synchronization

Data in the satellite downlink is organized in so called Channel Access Data Units (CADU) or frames. All CADUs have the same length of 1024 octets. To support the search for frame starts, the first part of each frame is an Attached Sync Marker (ASM) of 32 bit. The chosen bit pattern for the Terra satellite is 0001 1010 1100 1111 1111 1100 0001 1101 or in hexadecimal notation 0x1ACFFC1D. This sync marker is recommended for telemetry data channels by Consultative Committee for Space Data Systems (CCSDS). It will be used by other satellites like Aqua. Bit synchronization is considered done when the sync sequence is recognized. However, further checks can be done like

- Are there other sync markers 1024 octets before and after this position?
- Does the content of the frame passes any error test?
- Does the content of the frame makes sense to the next protocol layers?

Another task can utilize the attached sync marker pattern. In some demodulator configurations the polarity of bits may be inverted. The known pattern can be used to detect this condition and switch all the bits back. With ERSDEM2 and Terra this problem does not occur.

3.2.2 Frame Format

The next step is the extraction of MODIS packets, since the data stream until now may contain data from other on-board instruments or other telemetry data. Moreover the protocol overhead for the transport layer has to be removed.

As mentioned before, data from the spacecraft is organized in frames called Channel Access Data Unit (CADU). Each CADU has the constant size of 1024 byte. The first 32 bits of each CADU is the attached sync marker. The second part is exactly one Coded Virtual Channel Data Unit (CVCDU) of 1020 bytes.

Every CVCDU consists of 892 bytes Virtual Channel Data Unit (VCDU) and 128 bytes appended Reed Solomon forward error correction code. The purpose of virtual channels is the possibility to transmit several different data channels (for example different instruments) over one physical link (the satellite downlink).

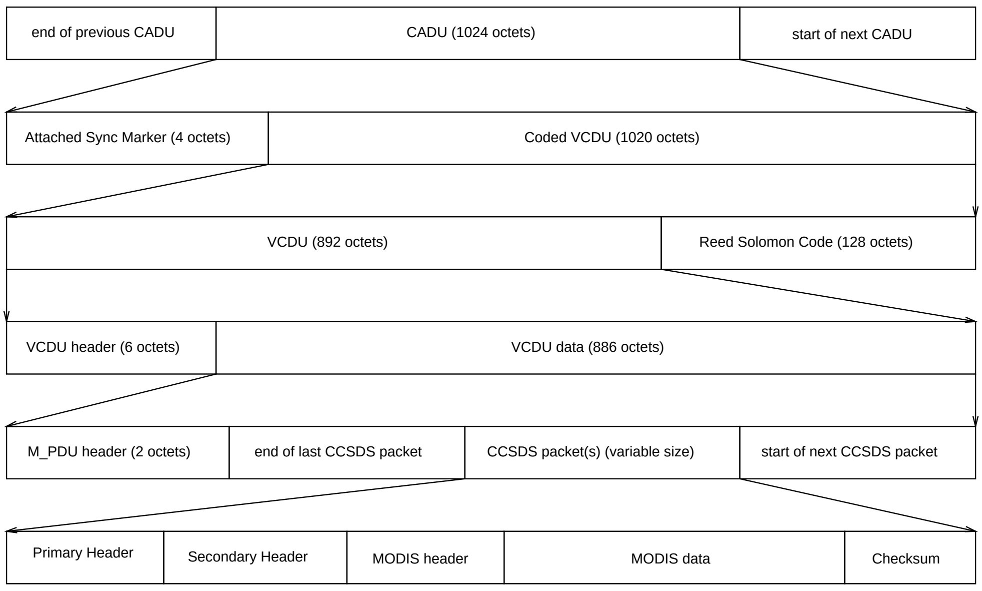
The virtual channel data unit has two main parts. The first part divided into the VCDU header and a M_PDU header. The second part consists of CCSDS packets.

The VCDU header contains the version number of the protocol which is 1 in the Terra and Aqua satellites. The next field states the globally unique spacecraft ID assigned by CCSDS. Terra has the ID 0x2a, Aqua will have the ID 0x9a [8].

The spacecraft identification field is followed by the number of the virtual channel. All virtual channel data units with the same spacecraft ID and the same channel number form one virtual channel. For the assignment of channel numbers see [19] page 61. A special channel ID is 0x3f which is used for empty VCDUs.

The next field is a count of virtual channel data units for each virtual channel. With this field it is possible to detect duplicated or missing parts in the telemetry data stream.

Figure 3: Downlink Packet Format



After the M_PDU header the CCSDS packets follow. Since these packets can have different lengths (for example day, night or fill packets), each packet carries its own length field and there is a pointer to the beginning of the first CCSDS packet in the M_PDU header. It is possible for a CCSDS packet to be split over multiple VCDUs. Even the header field of a CCSDS packet may be divided. It is also possible to have more than one packet per VCDU. It is the task of the raw to level 0 software to reassemble theses fragments.

To avoid repeating patterns besides the sync marker and ensure data transition density in the downlink stream, bits are XOR-ed with a pseudo random noise sequence

$$h(x) = x^8 + x^7 + x^5 + x^3 + x^1$$

[20]. This sequence is initialized to all One's at the start of every VCDU.

Each VCDU is followed by 128 octets Reed Solomon code for forward error correction to form a CVCDU.

The sequence of the two different codings varies between Terra and Aqua. In order to successfully decode frames the steps are for Terra

1. Decode Reed Solomon (RS) for entire VCDU (CADU without sync marker and RS code)
2. PN Decode VCDU but *not* the VCDU header

while for Aqua the steps are

1. PN decode CVCDU (*including* VCDU header and RS code)
2. Decode RS for entire VCDU.

3.3 The Program FrameSync

To perform the tasks described above, the program FrameSync was written at the University of Hawaii. The original version did not work very good on the data available at ITR so a few changes were done.

At first, the program starts to look for the sync marker pattern. The original version did this by bit shifting the whole data until the marker pattern appears at the beginning of a block. The changed variant shifts the sync marker through all possibilities and compares the orginal data with the shifted sync patterns, which is much faster. Internally the program works with 16 bit data, but the orginal version checked only for eight different bit shifts. Obviously 16 different byte shifts are possible with word size data. The search for the remaining eight contingencies was added.

Also added were a number of checks to prevent the output of invalid CCSDS packets. These include checks for packet size and correctness of header fields. To support the debuging a facility for dumping VCDUs was subjoined.

The program FrameSync was not tested with MODIS night packets since such data was not available. Further changes must be done when the Aqua satellite will be launched because the decoding sequence is slightly changed and a different spacecraft ID will be used.

3.4 Frame Synchronization with STPS

As described before, the program FrameSync as several disadvantages. Although there have been some improvements, a few bugs remain. For example, any damaged frame is still regarded as complete as long there is a attached sync marker at the beginning of it. The program ignores the Reed Solomon codes provided by the satellite. Another way of checking frames would be looking for the next sync marker, but FrameSync doesn't do this.

At the next stage, CCSDS packets have to be assembled from the VCDUs. (The format of the data stream is explained in [10].) If a CCSDS packet is not finished in the current frame, FrameSync waits for the second half. The next arriving fragment of a packet is appended to the first half, if the VCDU counter is the successor of the previous one. This works fine, because packages from all other Virtual Channels are discarded earlier. As mentioned above, the data used to complete packets is not checked. For example, if the pointer to the first CCSDS packet in the second frame is corrupt, a invalid packet would result. A length check is done, but MODIS night packets will fail this check.

Finally, the packets are written without checking the available CRC at the end of each MODIS packet.

That's why another way of doing this task was figured out. The software in the package `STPS_V3.41.tar.gz` from NASA Goddard Space Flight Center (GSFC) proved to be able to solve some of the problems. STPS stands for Satellite Telemetry Processing System.

The overall task is not done by a single program. In NASA's approach the programs *f2s*, *stpscc*, *dltp* and *pk* are involved. Additionally the tool *reformat* is necessary.

For all above programs except *reformat* and *f2s* it is necessary that a variable in the environment points to the base directory of STPS. In the csh or tcsh shell the needed command looks like
'`setenv STPS_LOCATION /home/.../stps`'. With the sh or bash shells it should be
'`export STPS_LOCATION=/home/.../stps`'.

The interaction between the different programs can be seen in figure 4 one page 18.

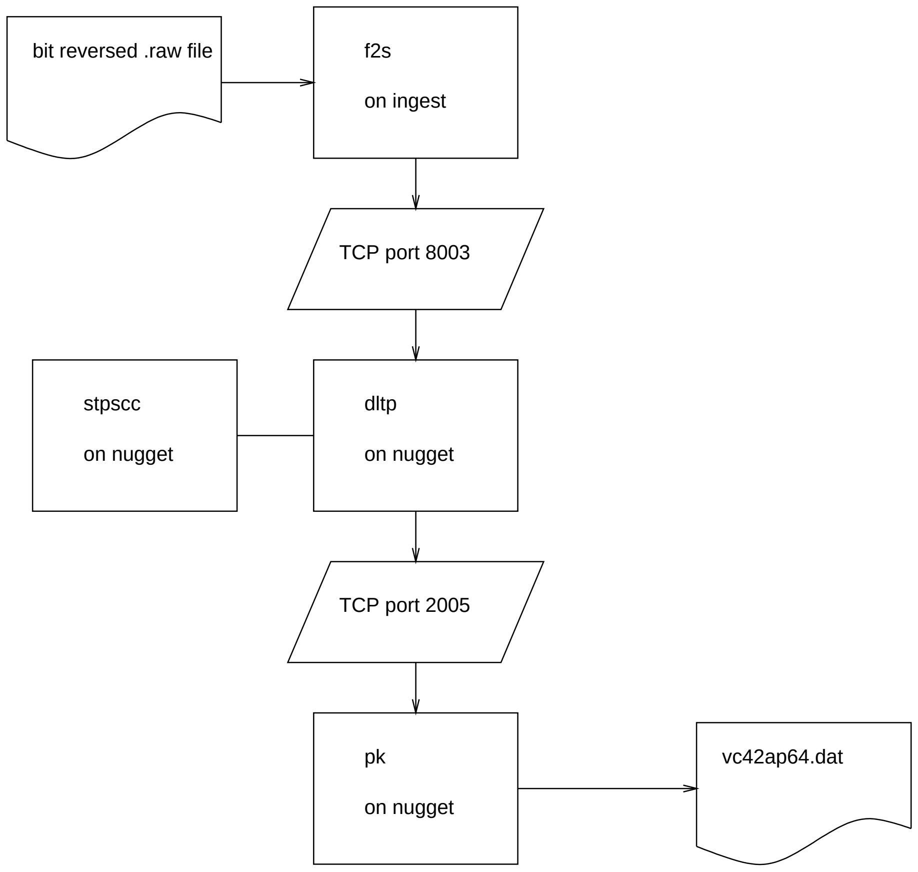
3.4.1 reformat

It was found that the `.raw` files available in ITR are byte-reversed. This is not a problem for the program FrameSync because the switch `-r` is made for this condition. However the program *dltp* cannot deal with such files so the have to be turned externally. Michael Sloman wrote the tool *reformat* which can do this task. If the bits in the files are ordered 1-2-...-7-8-9-10-...-15-16-17-..., after running through the tool the sequence is changed to 8-7-...-2-1-16-15-...-10-9-17...

The program acts as a filter, so a common way to call it is
`reformat <` *input file* `>` *output file*

Since the input files are not byte or word swapped and *reformat* reads the entire files byte by byte, it does not matter whether the executing computer is little endian or big endian machine.

Figure 4: Data flow in STPS



3.4.2 f2s

The tool *f2s* reads a file from disk, waits for a connection attempt on a specified TCP port and a hit from the keyboard. After transmitting the file via this connection, the program exits.

There are two reasons for using this tool. First the next stage programs *dltp* and *pk* can only run on big endian computers, so the data has to be transfered there somehow. Second the program *dltp* doesn't read from disk files, only from TCP ports.

The program's execution is controlled by a configuration file. An example of such a file is included in section F.1. For *f2s* it is necessary that the file contains a line starting with `F2S` and one starting with `*END*`. Any line not starting with a keyword is regarded as comment.

After the keyword `F2S` there have to be seven parameters. Their meaning are

- Name of the remote host, where *dltp* is running. It can be an IP address.
- TCP port number for the connection. 8003 is recommended.
- Block size for transmission. Small values increase transmission time. 32000 is large enough so larger numbers don't speed up the process any more.
- Wait time between packets in milliseconds. This value can be used to reduce system load on the target computer.
- Number of repeats. In this application it is not useful to set it to other values than 1.
- Mode of operation. A value of 0 means opening the socket in server mode, 1 client mode. Here only server mode is used.
- If the last value is set to 1 instead of 0, *f2s* outputs a few more status messages, which can be useful.

There are slightly different variants of *f2s*. The variants called *edos_f2s** transmit additional EDOS-header information in the TCP stream. Since *dltp* gets confused with this information, its use is not recommended. Other differences are whether *f2s* reads in the entire file at startup or reads it in in blocks as necessary. The author of *f2s* claims the first method would be faster. This could not be reproduced. The files are large compared to system memory, thats why the second method has been used.

3.4.3 stpscc and dltp

The reason why these two programs are described together in one section is their close interaction. The Channel Controller *stpscc* is responsible for reading in the configuration file, starting and stopping the child process *dltp* and printing out progress information.

It has some other capabilities like reading from DSP devices or communicating via UDP with a graphical user interface. These features have not been used here.

As usual, *stpscc* is configured via parameters on the command line and via a configuration file. On the command line there have to be four parameters. The first one names the configuration file, all the others deal with communication with the GUI and are not important here. Because all parameters are necessary for a successful start, a tiny shell script `runSTPS` is provided.

Once started, *stpscc* runs forever. The way to end the program is to send it a terminating signal either via control-C or via the kill command. If *stpscc* receives such a signal, it shuts all open connections down and kills *dltp*. Then the program exits.

The important program here is *dltp*. It receives via a TCP link the data from *f2s*. After the sync pattern is found several times at the expected place the LOCK state is entered. Then frames are forwarded to the EDOS output modules, where PN derandomization is done. In the output module the next step is the Reed Solomon decoding. A number of bit errors can be corrected. If there more errors than can be recovered, the frame is discarded.

After the frame was successful PN and RS decoded, it send via another TCP link to *pk*. In this data stream in front of each frame a EDOS header is send, which is necessary for *pk* to understand them.

An example of a working configuration file has been included in section F.2. The way *stpscc* reads in this file is not especially smart, thats why all lines have to appear and the sequence must not be changed. Comments in the file are not allowed.

In table 2 the explanation of some important lines can be seen. In a working file all lines have to be included, even if they have no meaning in this case, like the lines about the DSP process.

This software was developed on Sun machines using the GNU C compiler. On Linux computers it can be compiled only with changing the Makefiles and some source code lines. It doesn't run well on Linux machines.

Table 2: Some important settings in the configuration file of *stpscc*

VERSION 3.41	Necessary because the format of this configuration file changed between versions
error_log_file_dir ..	A directory where some progress information are stored.
status_log_file_dir ..	A directory where some progress information are stored.
dsp_process_up 0	Here no attempt is made to read directly from DSPs
input_buffer_size	Amount of data read in at once
input_data_type 0	Indicates data is not formated in any way. If this setting is not correct, *dltp* would not be able to read data.
check_input_block_crc 0	Frames are not protected by a CRC.
dltp_up 1	This line brings up the *dltp* process.
dltp_frame_data_to_file 0	Because data is send to the next stage it is not necessary to save it.
dltp_socket_host_machine	Where *f2s* is running
fs_bitReverseFrames 0	not the same as *reformat*
fs_pnDecodeCCSDS 0	PN decoding is done later in the EDOS module, which is configured at the end of the file.
fs_outputFramesState 2	meaning of state or mode: 0-Search, 1-Check, 2-Lock, 3-Flywheel[a].
fs_framesPerMode1 9	
check_frame_crc 0	Frames don't have CRC's
rsd_enabled 1	
rsdHeaderOnly 0	
EdosOutputServerHost	The computer *pk* is running on
EdosDiscardBadFrames 1	
EdosOutputSync 1	
TerraSpecificPNdecode 1	

[a]Flywheel mode in *dltp* is not comparable to flywheel mode in *FrameSync*

3.4.4 pk

The packet processor *pk* is used to extract CCSDS data packets from the stream of frames, reassemble them and separate them for different virtual channel and application ID. Here virtual channel 42 and application 64 (MODIS) in this channel are of interest.

The program resides in the subdirectory pkSERVICE_PROCESSOR. It started from the command line, usually with the switch `-s <configuration file>` set. If no filename is given, the default is `pk.config`. An example of the configuration file is given in section F.3.

For each channel and each application ID a separate file is written. All files are named `vc<VC ID>ap<app ID>.dat` so here the output file called `vc42ap64.dat` is the interesting one.

Normally *pk* writes EDOS headers in the output files. IMAPP is used in the next stage of data processing and doesn't understand them. That's why *pk* was patched to output only CCSDS packets. The output file can then be moved to the computer running IMAPP. Until now this is done via FTP, later maybe it should be done via NFS.

The program was implemented on a big endian machine and is dependent on this feature. It doesn't run on little endian computers.

3.5 IMAPP

IMAPP stand for International MODIS Processing Package and is derived from the operational MODIS processing software developed at NASA Goddard Space Flight Center. It consist of three main components.

The first part converts level 0 data to level 1a by examining the content of CCSDS packets. Then they get reformatted into an HDF file by the program L0_to_L1a. This program does a few checks on the data, for example it verifies the time ordering of the packets.

The second step is the geolocation of the data. Here a digital elevation model (DEM) is necessary to provide altitude information. The program `geolocate` can also do the geolocation without correction for the terrain which can save a little bit of processing time. Then the DEM is not necessary, a saving of hard disk space of about three gigabytes. In most cases the terrain correction is kept on, because the output is more accurate.

The last step is the calibration of the data the program `calibrate` and the production of the output HDF files with.

The calibrated data can be used for further level 2 processing. One example of data processing can be Dimple for classification (see appendix A).

In this work Matlab is used to produce a fire detection product.

4 Fire Detection

In the first part of this section the algorithms used for detecting any fires are described together with associated procedures like data input and output while in the next one (from section 4.4 onwards) the implementation of these algorithms is explained.

4.1 Algorithms for Preparing Fire Detection

Data in the HDF files produced by IMAPP is stored in 16 bit unsigned integers. The task is to assign a physical meaning to these values.

Each scientific data set in a HDF file can have associated attributes. For example, a long name of the data set can be stored. Here three attributes are important. To use the full possible range of 16 bit, a scale and a offset is used. The third attribute stores the physical unit.

The unit of radiance values is usually in $W \cdot m^{-2} \cdot \mu m^{-1} \cdot \text{steradian}^{-1}$. If scale s, offset o and the data x from the HDF file are known, the radiance B can be computed as

$$B = (x - o) \cdot s \cdot \frac{W}{m^2 \cdot \mu m \cdot \text{steradian}}.$$

The same formula applies for reflectance values with the modification that reflectance does not have a physical unit associated.

But most algorithms for fire detection do not work directly with radiance values, they need data in means of brightness temperature. Brightness temperature is the temperature where a black body would emit the same amount of radiation for the given value.

To convert radiance values to brightness temperature, the inversion of Planck's formula is used as found in [12] or [2].

$$T = \frac{C_2/\lambda}{\ln\left(C_1/(\lambda^5 \cdot B) + 1\right)}$$

where

C_1	first radiation constant $C_1 = 2\hbar c^2 = 1.1910439 \cdot 10^{-16} \quad Wm^{-2}$
C_2	second radiation constant $C_2 = \hbar c k^{-1} = 1.4387686 \cdot 10^{-2} \quad mK$
T	brightness temperature (K)
B	radiance $(\frac{W}{m^2 \cdot m \cdot \text{steradian}})$
λ	center wavelength of channel (m)
$\hbar$	Planck constant
c	speed of light
k	Boltzmann constant

Table 3: Possible values for land / sea mask

Value	Meaning
0	Shallow Ocean (Ocean less than 5km from coast OR less than 50m deep).
1	Land (not anything else).
2	Ocean Coastlines and Lake Shorelines.
3	Shallow Inland Water (Inland Water less than 5km from shore OR less than 50m deep).
4	Ephemeral (intermittent) Water.
5	Deep Inland Water (Inland water more than 5km from shoreline AND more than 50m deep).
6	Moderate or Continental Ocean (Ocean more than 5km from coast AND more than 50m deep AND less than 500m deep).
7	Deep Ocean (Ocean more than 500m deep).

If B is still in $W/m^2 \cdot \mu m \cdot$ steradian a factor of 10^6 has to be inserted like in [12] or appendix C.2.

4.1.1 Evaluation of the Land / Sea Mask

The land/sea mask is provided by IMAPP in a 1km resolution and can have the values listed in table 3 as mentioned in [22]. Only pixels with a land/sea mask value of 1 (land) or 2 (land with coastline / shoreline) should be tested for fires, all other pixels are ignored.

4.1.2 Creation and Evaluation of the Snow Mask

Snow has a very high reflectance in many bands and possibilities are to confuse fire pixels with pixels containing snow. Even if snow and ice is not common in most parts of Australia, the Snowy Mountains are covered in most of the years.

To find snow pixels in the image, the Normalized Differential Snow Index (NDSI) is utilized. This index has been tested with data from Landsat and data from MODIS instruments. It is the basis of [17] and [13]. The thresholds used in this work are based on [13].

The NDSI works similar to the NDVI (normalized differential vegetation index), but uses the bands centered at 555 nm and 1640 nm[1] instead of red

[1]MODIS channels 4 and 6

and near infrared:

$$NDSI = \frac{\rho_{0.555} - \rho_{1.64}}{\rho_{0.555} + \rho_{1.64}}$$

The snow mapping consists of three tests.

- $NDSI > 0.4$
- $\rho_{0.858} > 11\%$ (band 2)
- $\rho_{0.555} > 10\%$ (band 4)

The two latter tests prevent very dark pixels to be mapped as snow. Only if all three tests are passed the algorithm considers the area as snow covered.

The use of the created snow mask is that on snow pixels there can't be any fire.

4.1.3 Cloud Detection

Similar to snow, clouds have large reflectance values and if no care would be taken, they would be regarded as fires. In [1] a lot of effort is undertaken to classify different type of clouds like thin cirrus, clouds over snow fields or high thick clouds. Similar in [4] at least eight different test for different cloud types are done. Here this differentiation is not necessary and so a different approach was chosen.

As mentioned, clouds reflect many wavelengths very good. So again three test are done:

- $\rho_{0.488} > 95\%$ (band 10)
- $\rho_{0.531} > 95\%$ (band 11)
- $\rho_{0.551} > 95\%$ (band 12)

If and only if all three tests are passed, a pixel is declared cloudy and no fire detection is done. The test was validated with day images from MODIS only.

4.2 Fire Detection Algorithms

There are a different approaches to detect fires in remote sensing data. The simplest form compares the brightness temperature around $4\mu m$ with a given threshold. This works well if the remaining image is relatively cool (especially at night) but this is not the case for the area in South Australia.

Other works like [6] use a ratio between the AVHRR $3.7\mu m$ channel and the background. Kudoh describes in [14] the use of normalized difference indices between NOAA AVHRR channels 3A, 2 and 1.

In [16] also two traditional threshold test are employed plus a difference between NOAA channels 3 and 4.

Each technique has its disadvantages. Fixed threshold methods have problems with the different types ecosystem found in large images from MODIS. Additionally these fixed threshold vary seasonally. Some other methods like the neural network approach in [18] need much computation time not available for near real time applications like this work.

Here a algorithm similar to the one in [15] is used. It contains difference tests, comparisons with backgrounds and threshold tests.

The simplest test is a fixed threshold test. If

$$T_{3.9} > 360K \tag{1}$$

then a fire is assumed regardless of the results of other tests. Even if the fire on the ground is much hotter, the effective brightness temperature rarely reaches this value. This has different reasons. The spatial resolution in the channels used is 1 km, so most fires are smaller than a pixel. Second the radiance is scattered in the atmosphere and often obstructed by the smoke plume from the fire.

Next the brightness temperature is compared to another value

$$T_{3.9} > 325K \tag{2}$$

but this is only an intermediate result. Then the difference between Channel 21 ($3.9\mu m$) and channel 31 ($11\mu m$)

$$\Delta T = T_{3.9} - T_{11}$$

is computed and compared to the fixed threshold

$$\Delta T > 25K \tag{3}$$

With the background values $T_{3.9b}$ and ΔT_b and their standard deviations $\delta T_{3.9b}$ and $\delta \Delta T_b$ two further tests are done

$$T_{3.9} > T_{3.9b} + 4\delta T_{3.9b} \tag{4}$$

$$\Delta T > \Delta T_b + 4\delta \Delta T_b \tag{5}$$

The values are compared with their background. If the difference is larger than four times the standard deviation the test states a possible fire on the given location.

If the following logical combination

$$Test1 \vee ((Test2 \vee Test4) \wedge (Test3 \vee Test5))$$

is true then the pixel is declared to be a fire pixel. So it has to pass one of the $3.9\mu m$ tests and one of the ΔT tests. Additionally, if it passes the big threshold test (1) it is declared a fire pixel regardless of the outcome of all other tests.

For night passes which may be recorded in the future, other threshold values are suggested. The values of 360K, 25K and 325K should be replaced, according to [15] with 330K, 10K and 315K. Since until now no night pass is available, these values couldn't be verified in any way.

4.3 Algorithms using the Results of Fire Detection

With the results of the fire detection at hand, next steps are the visualization of these results. The goal is first a list of all fires found and second a undistorted map with fire pixels marked.

4.3.1 Map Projection

As explained in section 2.7, data from MODIS is subject to the Bow Tie effect and a few other artifacts. To remove these effects and to put the data into common format, the rectangular latitude/longitude projection was chosen. With this projection all latitude and longitude lines are assumed parallel and of equal distance. Of course this is not true so this projection introduces distortions (as every projection from a sphere to a plane does, see [27]) into the image. The further apart from the equator, the more expanded the data get. To balance out this effect a little bit the following assumptions are made:

- The distance between latitude lines (set of points with the same integer latitude value) is 111.2km. This not exact for every point on earth but sufficient for this application.
- The distance between longitude lines is

  $$111.2km \cdot \cos\phi$$

  for the entire image. This also assumes a spherical earth but has a sufficient accuracy here.
- ϕ was chosen to be 34.86 degrees South as this is the position of the receiving dish in Mawson Lakes/Adelaide.

- The size of the smallest pixels in the nominal 1km resolution is exactly 1 km x 1 km. This is usually only true for pixels along the nadir line.

These assumptions have the consequence that the image gets compressed for parts north of Adelaide and expanded for parts south of Adelaide. Another result of trading all images with the same assumptions is that these images can fit together to form a bigger map if desired.

With the now known distance between latitude and longitude lines a grid can be constructed based on the maximum latitude and longitude values found in the geolocation data. Each pixel in the constructed grid has a height of

$$(111.2\frac{km}{deg})^{-1} \cdot 1\frac{km}{pixel} = 0.0089928\frac{deg}{pixel}$$

and a width of

$$(111.2\frac{km}{deg})^{-1} \cdot 1\frac{km}{pixel} \cdot \cos(\phi_{Adelaide}) = 0.010959\frac{deg}{pixel}.$$

These values are constant for all images. The maximum shift between two different images is one pixel because of different image origins. Despite of this one pixel (or less) shift different images fit together.

4.3.2 Smoothing

The drawback of the described projection is that many pixels in the resulting image stay black because the image element size at the left and right edge is much larger than 1km. In fact a image element can grow based on the 55 degree maximum view angle of MODIS up to 5x2km. Since only the one recorded element gets projected in the final grid, many spots in this grid remain black. These black spots have to be filled with data in order to get a viewable image.

Two different ways of filling these black points have been tried. The first approach was the use of median rank ordering filters for each color band. For each pixel to be filled the median filter considers all its neighbors according to the filter size, sorts them in value order and selects the one in the middle. For larger filter sizes this apparently introduces errors because pixels far away from the one to be filled can largely influence the result. For small filter sizes the number of candidates often gets even (like 2). Since the filter now can't take the middle value the images gets either too dark (if the filter choose value 1) or too bright (if the filter tries value 2 in this case).

The second variant of pixel filling was with a averaging filter. Here the sum of all non zero pixels in the filter size is divided by their count. The disadvantage of larger filter sizes is like above the influence of extreme pixels (like clouds) spread over a large area. If a smaller filter size is chosen, many dark pixels remain.

The remaining black pixels can be eliminated by reapplying the filter. The best results have been found with an averaging filter of size 3x3 applied twice. Smaller square shaped filter sizes do not make sense since a 1x1 filter doesn't change anything and a 2x2 filter shifts the image in one direction. The next larger odd sized filter (5x5) already introduces many more errors.

4.4 Implementation of Fire Detection

For the realization of the described algorithms Matlab 6 Release 12 was the chosen language. This has several reasons:

- Matlab has a good support for all needed file formats, particularly the input format HDF and the output format TIFF.
- Algorithms are easy to formulate in Matlab. The written programs are easier to understand than similar programs in for example C.
- The resulting programs are highly platform independent and can run on every platform supported by Matlab.
- The execution speed is very good while doing matrix operations and sufficient in all other cases.
- Many common exceptions are caught by Matlab. For example a division by zero is not uncommon while computing the NDSI. This causes only a suppressible warning message and the result NaN (not a number) and not a runtime error with program abort.

A disadvantage of Matlab is its demand to store every data for computations in 64 bit double values. This is especially memory inefficient for 1 bit logical arrays.

All computations from the end of the IMAPP run are mainly done in one Matlab function. The aim of this section is to describe the implementation of the algorithms mentioned above and the running of the Matlab script `firedetection` in some details.

A sample run of the script produces output which looks similar to the one in figure 5. There the main parts of the implementation can be seen. The function consists of the following parts:

1. Reading in all necessary data, i.e. importing parts of the HDF files

2. Executing sanity checks like detection of clouds and water
3. The actual fire detection
4. computation of the map
5. And finally the output of the results.

Now follows a more detailed look at the source code of the function and its execution. The entire source code of the script is in appendix C.1.

The first main task is the import of necessary data. Data from IMAPP is stored in HDF (Hierarchical Data Format) files. For a better description of the file format see section 2.6.3 and [25]. Since Matlab has an excellent support for the HDF format, the functions for reading data (`sds_read.m` in appendix C.4) and associated attributes (`attribute_read.m` in appendix C.3 can be kept short.

For the different algorithms a number of bands and some meta information is necessary. So the script reads in the MODIS bands 1, 2, 3, 4, 6, 10, 11, 12, 21 and 31. Additionally the meta information latitude, longitude and land/sea mask are read. Error checking is done where applicable.

The algorithm for firedetection needs data in terms of brightness temperature, but values in the HDF files are stored as radiances. To automate the conversion from radiance to brightness temperature the function `radiance2teff.m` was written. Calling syntax is

```
result=radiance2teff(radiance,lambda)
```

where `radiance` is expected in $W/m^2 \cdot \mu m \cdot$ steradian as found in the HDF files and `lambda` should be in μm. After successful completion `result` is returned in Kelvin. The parameter `radiance` can be a matrix, so many values can be converted at once.

The source code is found in appendix C.2. The usual Matlab help facility via `help radiance2teff` is supported.

The next step in the fire detection script is the elimination of apparently bad data. This includes reflectance values greater than one, brightness temperatures greater than the saturation values of the sensor, values where `radiance2teff` computed invalid results or where the bad data marker was inserted by IMAPP (-999 for latitude/longitude values).

The evaluation of the provided land/sea mask is done as described above, only values of 1 or 2 are regarded as possible places for fires. The elegance of implementing such algorithms in Matlab can be seen in lines like

```
 land = (landseamask==1 | landseamask==2);
```

which creates a logic array of the necessary size containing true for all land pixels and false for all others.

Figure 5: Sample output from the script `firedetection`

```
>> firedetection(FILENAME);

read data from disk
 meta information
 1km emissive
 1km reflective
 250m reflective
 500m reflective
reading finished
converting radiances to temperature values
 4um
 11um
looking for bad data
eliminating bad data
evaluating land/sea mask
detecting snow
detecting clouds
detecting fires
 computing backgrounds
 testing
 finding
 number of fire pixels detected: 78
creating rgb image
producing rectangular map
 calculating vectors and copying data
  one degree latitude is 111.2 pixels high
  one degree longitude is 91.2453 pixels wide
 smoothing (1/2)
  finding black spots
  count pixels we have got
  filtering
   red
   green
   blue
 smoothing (2/2)
  finding black spots
  count pixels we have got
  filtering
   red
   green
   blue
 painting fires
writing image
writing ASCII output
all done

>>
```

A problem during snow detection (in fact during computing the NDSI) arises when both values are zero. Then the divisor becomes zero and a division by zero warning occurs. This warning can be and is suppressed. The invalid result is no problem because when both band values are zero then there is definitely no snow at this point. Additionally one of the other two tests fails too.

The cloud detection works straightforward except at the end, the found cloud mask is broadened by one pixel. This is done by convoluting the cloud mask with a 3 by 3 matrix containing only ones. Because the cloud mask is a array of logic values, the filter could be described as a 3 by 3 logic OR filter. The effect is a smother cloud mask including cloud edges.

For the fire detection the computation of the pixels background is needed. This is achieved with the filter

$$\frac{1}{40}\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

The problem with convolution filters is always at the edges. That's why the edges are excluded from further processing.

The standard deviation of each band is computed for the whole image and then background values are compared to the pixels. All threshold tests are done with day time values. A night time test was not done at all.

The rest of the function deals mainly with the computation of the rectangular latitude/longitude map.

The values in the visible bands have to be compressed into the 8×3 bit in order to display them. To further improve the image, a clipping of higher values is done in the subfunction `stretch`.

The latitude and longitude values have to be converted to X and Y positions in the final map. This is the task of the function `cstretch`, which takes the gridding factor as an input. The gridding factor is 111.2 kilometer for latitude values and $111.2km \cdot \cos(\phi_{Adelaide})$ for longitude values. When the conversion is finished, the red, green and blue pixel values are copied into the map.

Not all pixels in the final map get filled with this procedure. The remaining pixels are filled with averages of their neighbors.

After the detected fire pixels are marked with red spots in the smothed map, the image is written to disk and the reporting text file is generated. At this point the function returns.

5 Conclusions

All programs mentioned and all algorithms described are called and run in the necessary sequence by the small shell script in appendix E.3. This script does all work from the recorded raw file to the fire detection product.

In this script the program FrameSync is used instead of STPS. This decision was made to simplify the process and keep all computation on one place. Additionally it proved difficult to coordinate the many different programs of the STP System, sometimes keystrokes are necessary. This is not desirable in an automatic system. Because `slicer` eliminates most of the bad data already, the lack of error checking in FrameSync is not that adverse.

The results of the fire detection algorithms have been empirically validated in a number of ways:

- At first a visual inspection of the resulting map was done. All fires found are on land and not obstructed by clouds. All fires were near either a visible smoke plume or a visible fire scar. However some smoke plumes don't have a fire pixel near them. This has several reasons. Some fires are just too small to be picked up by the 1km x 1km resolution of MODIS. In some cases the smoke plume itself prohibits the detection of the causing fire by either being detected as a cloud or just obstructing the view to the ground.

- Next step was the comparison to similar pictures published by NASA in [24]. Since the available images were four days apart, only a very rough comparison was possible, but it showed a similar configuration of fire pixels. Some fires have advanced a distance, others were extinguished or ignited.

- Last the data was compared to the data published daily by DOLA [9] on their web site. There the NOAA-AVHRR data is used to detect fires. The comparison looks promising and shows many similar results. Differences are caused by the different recording time and the use of another sensor. A statistical analysis was not done because of lack of enough data.

All validations done show the results are very reasonable. The lack of more data prevented a more statistical evaluation of the correctness of the algorithm.

A sample output of the work can be see in figure 6. The found fires in this figure are listed (among others) in figure 7. The image was recorded on 10/08/2001 and shows an area near Gulf of Carpinteria in the Northern Territory. Fire scars and smoke plumes can easily be spotted.

Figure 6: Picture of detected fires

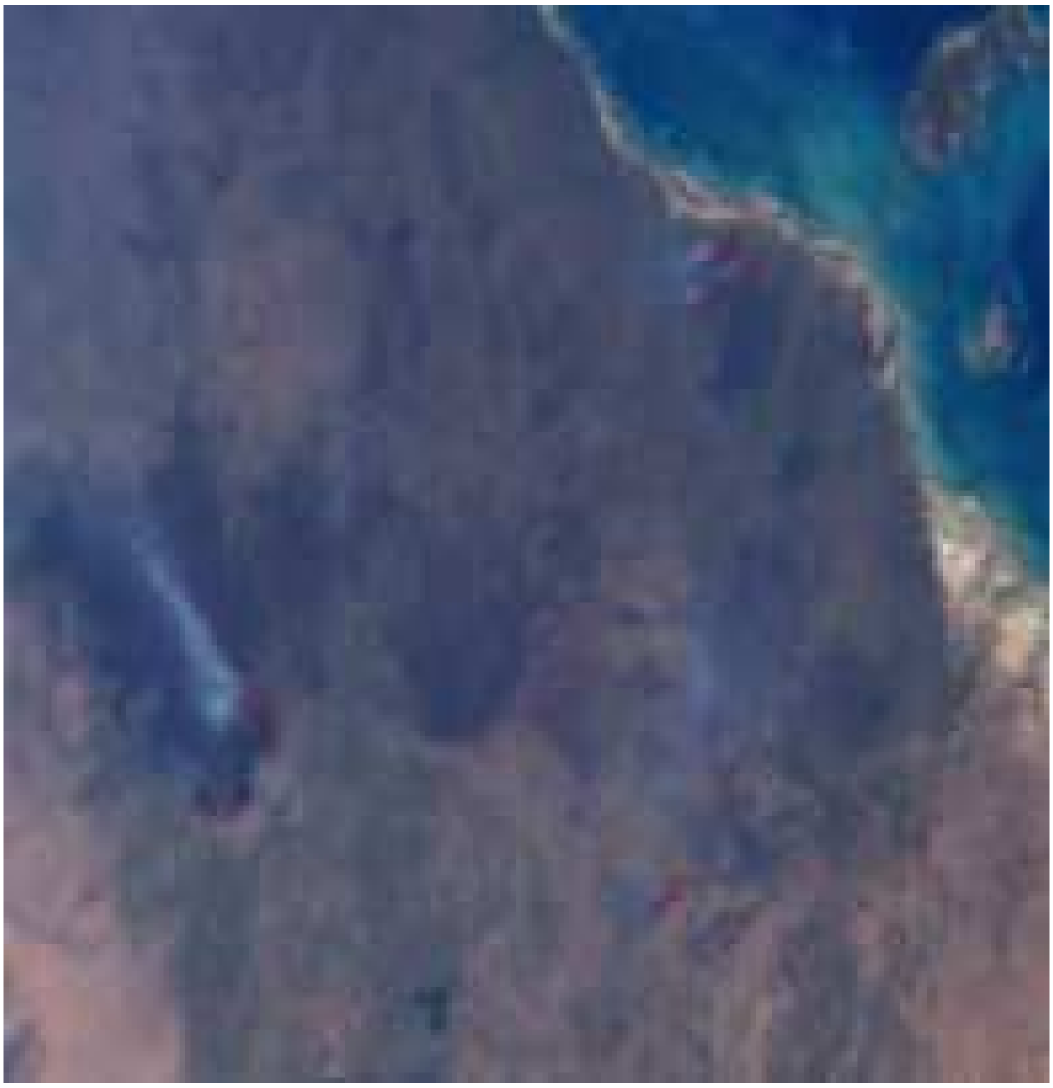


Figure 7: List of detected fires

```
# Results of firedetection
# input files:
#      /dd1/IMAPP_RUN/ter10aug58-174.geo.hdf
#      /dd1/IMAPP_RUN/ter10aug58-174.1000m.hdf
# output files:
#      /dd1/IMAPP_RUN/ter10aug58-174.fires.tiff
#            image range: -20.9236/133.1496 to -14.3969/155.9702
#            pixelhigh:  0.0084998 degrees latitude
#            pixelwidht: 0.016104 degrees longitude
#      /dd1/IMAPP_RUN/ter10aug58-174.fires.txt (this file)
# starting time: 25-Nov-2001 11:31:18
# finishing time: 25-Nov-2001 11:33:46
#
# the following fires were found:
# latitude  longitude    posit-x    posit-y
 -16.87312  135.21036        291        128
 -16.87901  135.21593        292        128
 -16.88553  138.45078        293        329
 -16.88549  138.44351        293        329
 -16.88902  138.46284        293        330
 -16.88893  138.47014        293        330
 -16.89471  138.42897        294        328
 -16.89810  138.44839        294        329
 -16.89806  138.44115        294        329
 -16.90503  135.20378        295        128
 -16.90158  138.46049        295        330
 -16.90147  138.46777        295        330
 -16.90994  138.57751        296        337
 -16.90902  138.58495        296        338
 -16.91895  138.55623        297        336
 -16.91815  138.56363        297        336
 -16.92238  138.57518        297        337
 -16.92145  138.58260        297        337
 -16.97014  138.40166        303        326
 -16.97005  138.40747        303        326
 -16.97369  138.42108        303        327
 -16.97346  138.42691        303        328
 -17.05182  141.13303        312        496
 -17.05598  141.09351        313        493
 -17.74360  136.74628        394        223
 -17.75962  136.76088        396        224
 -17.76906  136.73309        397        223
 -17.78312  136.73033        398        222
 -17.78768  136.75540        399        224
 -17.79222  136.78035        399        225
 -17.79673  136.80521        400        227
 -17.81071  136.80246        402        227
 -17.96402  136.60155        420        214
 -17.96860  136.62700        420        216
 -17.97316  136.65234        421        218
 -18.14500  138.27626        441        318
 -18.14830  138.29526        441        320
 -18.15743  138.27394        442        318
 -18.16072  138.29294        443        319
 -18.16398  138.31187        443        321
```

6 Recommendation

There are a number of fields where further work can be done. These include

- Attempt to forecast fire intensity like in [21]
- The mapping has disadvantages. Especially when a temporal analysis is desired, a more accurate gridding of the data is necessary. Works like [28] show how this can be done.
- It is possible to detect the amount of smoke to draw conclusions about the fires. Zhanqing Li et.al. suggested in [18] the use of neural networks to distinguish between smoke and clouds.
- Fire scar detection is useful for the estimation of the amount of burnt area and the amount of produced CO_2. In [5] this is shown with AVHRR data.
- The program `slicer.c` should be modified to slice even when no bad data is found at certain file sizes like every 400 megabyte. This would prevent some lack of memory problems and speed up processing. Additionally the storage of images may be easier.
- Further validation must be done. Especially DOLA [9] should be a good resource for this work.
- When Aqua will be launched a few adjustments must be done to the FrameSync program to adopt changes in
  - sequence in PN and RS decoding
  - and different spacecraft ID.
- Processing of night images was not tried at all and may promise good results in fire detection (but not in imagery). For this to work some minor changes to FrameSync are necessary. The function `firedetection` also has to be changed a bit because some bands are not available at night.

A Dimple

Dimple is a digital image processing system from Process Software Solution Wollongong, Australia. It is designed as a general image manipulation and processing system but has some interesting features specially for remote sensing and satellite imagery.

The general features include some image transformations like from RGB to grey scale images, 3D plots or different image ratios. Since normalized ratios are supported, the latter one can be used for producing indices like NDVI or NDSI.

The image enhancement part has the standard filters including convolution and rank order filters. Special filters are for example the de-striping filters for satellite images.

The built-in filters can be enhanced with the Image Operation Language IOL which is a simple programming language for image manipulation.

The rectification, registration and resampling algorithms provided work all with ground control points which have to be supplied by the user. This is a manual process and was therefor not seen as an option for this project. The same is true for the classification algorithms. They have to be trained manually.

The main disadvantage in conjunction with MODIS data is the inability of Dimple to read HDF files. To overcome this Matlab was used to import HDF files and export them in a format Dimple can read. Appendix D.2 contains the source code of this Matlab function. Data is exported in binary files containing no other information than the actually calibrated data from IMAPP. To import these files in Dimple, Dimple has to know how to read them, so the user has to supply for example the resolution of the image and the number of bands to import.

Dimple can read such binary files with different data types after the user has supplied this information. For example it is possible to transfer latitude/longitude data as real numbers and integer data as uint16.

B Glossary

ASM Attached Sync Marker; beginning of every CADU

ASTRA Automatic Satellite Tracking Research Antenna; receiving facility in ITR

AVHRR Advanced Very High Resolution Radiometer; Instrument on board NOAA satellites

Byte Eight bits, used synonymously with octet.

CADU Channel Access Data Unit; consists of ASM and CVCDU

CCSDS Center for Computational Science and Advanced Distributed Simulation

CVCDU Coded Virtual Channel Data Unit; consists of VCDU and Reed Solomon Code

DEM Digital Elevation Model

HDF Hierarchical Data Format; used for data storage in level 1a and level 1b

ITR Institute for Telecommunications Research

MODIS Moderate Resolution Imaging Spectroradiometer; Instrument on board Terra and Aqua satellites

M_PDU Multiplexed Protocol Data Unit

NCSA National Center for Supercomputing Applications

NDSI Normalized Difference Snow Index

NDVI Normalized Difference Vegetation Index

NOAA National Oceanic and Atmospheric Administration; part of U.S. Department of Commerce

Octet Eight bits, used synonymously with byte.

STPS Satellite Telemetry Processing System; Software from GSFC

VCDU Virtual Channel Data Unit; transport entity for CCSDS packets

word sixteen bits of data

C Matlab Source Files for Fire Detection

In this section all Matlab files used while firedetection are listed. Some additional functions, which are not necessary but sometimes helpful, can be seen in appendix D.

C.1 `firedetection.m`

```
function f=firedetection(filename);

% FIREDETECTION detect fires using data in `filename`
%
% fire detection is a longer script
%  for documentation, please refer to
%  the source code and
%  Kendy Kutzner:
%   "Processing MODIS Data for Fire Detection in South Australia",
%   2001, ITR, Adelaide

% change history
% 08/2001 to 10/2001 written by kendy kutzner
% 11/2001 documented by kendy
% 12/2001 speed up by vectorizing mapping by kendy
% 12/2001 further documentation by kendy

% let's start
starttime=now;

% no failure until now
% most common errors are catched.
% of course not all.
% probably the the most seriuos not catched
% error is out of memory. In this case matlab
% aborts the function. This script tries to clear memory
% as early as possible.
f=0;

% read all the data
disp('read data from disk');
% at first, we need some information about the data in
% the .1000m.hdf file
disp(' meta information');
filename1=[filename '.geo.hdf'];
% for obvious reasons we need these:
[latitude,stat1]=sds_read(filename1, 'Latitude');
[longitude,stat2]=sds_read(filename1, 'Longitude');
% there are no fires on water, so make use of
% the land/sea mask
[landseamask,stat3]=sds_read(filename1, 'Land/SeaMask');
% to have better visual control when needed all
% data is transposed so it can be viewed with
% imagesc(bandX)
% additionally, the imwrite() function wants the
% data in this direction to produce images in the
% 'right' direction
latitude=double(latitude');
longitude=double(longitude');
landseamask=landseamask';
```

```
% catch some errors
if (stat1~=0 | stat2~=0 | stat3~=0)
    f=-1;
    disp('Can''t read Latitude/Longitude or Land/Sea Mask');
    return;
end

% now we read in all the data bands we need.
disp(' 1km emissive');
filename1=[filename '.1000m.hdf'];
bandname='EV_1KM_Emissive';
[emissives,stat1]=sds_read(filename1, bandname);
[scales,stat2]=attribute_read(filename1, bandname, 'radiance_scales');
[offsets,stat3]=attribute_read(filename1, bandname, 'radiance_offsets');
[names,stat4]=attribute_read(filename1, bandname, 'band_names');
if (stat1~=0 | stat2~=0 | stat3~=0 | stat4~=0)
    f=-1;
    disp('Can''t read Emissive Bands');
    return;
end
offsets=double(offsets);
scales=double(scales);
% data in .hdf files is stored with scale and offset
% to make better use of 16bit data fields.
% We need the actual values so convert them.
band21=((double(emissives(:,:,2))-offsets(2))*scales(2))';
% transpose the arrays for the reason above
band31=((double(emissives(:,:,11))-offsets(11))*scales(11))';
% these arrays tend to be quite big, so free memory if we can
clear emissives;
% if lack of memory is still a problem, a 'pack' statement can
% be inserted on the critical points.

disp(' 1km reflective');
filename1=[filename '.1000m.hdf'];
bandname='EV_1KM_RefSB';
[reflectives,stat1]=sds_read(filename1, bandname);
% for reflective values, different scales and offsets apply
[scales,stat2]=attribute_read(filename1, bandname, 'reflectance_scales');
[offsets,stat3]=attribute_read(filename1, bandname, 'reflectance_offsets');
[names,stat4]=attribute_read(filename1, bandname, 'band_names');
if (stat1~=0 | stat2~=0 | stat3~=0 | stat4~=0)
    f=-1;
    disp('Can''t read Reflective Bands (1km)');
    return;
end
offsets=double(offsets);
scales=double(scales);
band10=((double(reflectives(:,:,3))-offsets(3))*scales(3))';
band11=((double(reflectives(:,:,4))-offsets(4))*scales(4))';
band12=((double(reflectives(:,:,5))-offsets(5))*scales(5))';
clear reflectives;

disp(' 250m reflective');
filename1=[filename '.1000m.hdf'];
bandname='EV_250_Aggr1km_RefSB';
[reflectives,stat1]=sds_read(filename1, bandname);
[scales,stat2]=attribute_read(filename1, bandname, 'reflectance_scales');
[offsets,stat3]=attribute_read(filename1, bandname, 'reflectance_offsets');
[names,stat4]=attribute_read(filename1, bandname, 'band_names');
if (stat1~=0 | stat2~=0 | stat3~=0 | stat4~=0)
    f=-1;
    disp('Can''t read Reflective Bands (250m)');
    return;
end
offsets=double(offsets);
scales=double(scales);
band1=((double(reflectives(:,:,1))-offsets(1))*scales(1))';
band2=((double(reflectives(:,:,2))-offsets(2))*scales(2))';
clear reflectives;

disp(' 500m reflective');
filename1=[filename '.1000m.hdf'];
bandname='EV_500_Aggr1km_RefSB';
[reflectives,stat1]=sds_read(filename1, bandname);
[scales,stat2]=attribute_read(filename1, bandname, 'reflectance_scales');
[offsets,stat3]=attribute_read(filename1, bandname, 'reflectance_offsets');
[names,stat4]=attribute_read(filename1, bandname, 'band_names');
if (stat1~=0 | stat2~=0 | stat3~=0 | stat4~=0)
    f=-1;
    disp('Can''t read Reflective Bands (500m)');
    return;
end
offsets=double(offsets);
scales=double(scales);
band3=((double(reflectives(:,:,1))-offsets(1))*scales(1))';
band4=((double(reflectives(:,:,2))-offsets(2))*scales(2))';
band6=((double(reflectives(:,:,4))-offsets(4))*scales(4))';
clear reflectives;
clear scales offsets bandname filename1
% hopefully now we got all the data we need
disp('reading finished');
% correctness of values until now verified with HDFLook
% from now on, we are on our own

disp('converting radiances to temperature values');
% calculate the middle of the channels on the fly
disp(' 4um');
band21=radiance2teff(band21,(3.989+3.929)/2);
disp(' 11um');
band31=radiance2teff(band31,(10.780+11.280)/2);

disp('looking for bad data');
% if the conversion above produced complex values,
% the argument to log() was negative. Something
% is seriously wrong, we want to get rid of this.
imags=imag(band21) | imag(band31);
% the 4um channel saturates at 500K, the 11um channel at 400K
% so all values above are bad
% same to reflective bands: values above 100% are bad
baddata=find(band1>1 | band2 >1 | band3>1 | band3>1 | band4>1 | ...
             band6>1 | band21>500 | band31>400 | imags | ...
             latitude==-999 | longitude==-999);
clear imags;
disp('eliminating bad data');
for i=[1 2 3 4 6 10 11 12 21 31]
    istr=num2str(i);
    evalstring=['band' istr '(baddata)=0;'];
    eval(evalstring);
end
% special case for lat/long, because we need the
% min() and max() function later. If we would insert
% zeros, one of them would get confused.
% min() and max() take care of NaNs, so we can use it
```

```
latitude(baddata)=nan;
longitude(baddata)=nan;

disp('evaluating land/sea mask');
% we want only "land" or "land with coastline"
land=(landseamask==1 | landseamask==2);
clear landseamask ;

disp('detecting snow');
%next line may produce division by zero warning.
% this is ok, so turn it off
warning off;
ndsi=(band4-band6)./(band4+band6);
% we want to hear warnings again
warning backtrace;
% this is the snow detection algorithm
snow=((ndsi>.4 & band2 > .11) & band4>.1);
snowpos=find(snow & land);

% next to the clouds
disp('detecting clouds');
cloud=((band11> 0.95*max(band11(:))) & ...
       (band10> 0.95*max(band10(:))) & ...
       (band12> 0.95*max(band12(:))));
% the above cloud detection algorithm is not the best
% To get all the cloud edges, we broaden the cloud mask
% with a smoothing filter
cloud=conv2(cloud,ones(5),'same');
% bring cloud mask to logic array again
cloud=cloud & 1;
cloudpos=find(cloud & ~(snow & land));

% now to the main part
disp('detecting fires');
% at first we construct a background filter
backgroundfilter=ones(7);
% we are only interested in the surrounding
backgroundfilter([3 4 5],[3 4 5])=0;
% normalize the filter so it doesn't disturb data
backgroundfilter=backgroundfilter/sum(backgroundfilter(:));

disp(' computing backgrounds');
% it proved convinient to mark bad data with NaN
band21(band21==0)=nan;
band31(band31==0)=nan;
t41=band21-band31;
t4b=conv2(band21,backgroundfilter,'same');
t41b=conv2(t41,backgroundfilter,'same');
% by filtering, the edge of each matrix becomes invalid
% so we construct a invalid mask
invalid=zeros(size(t4b));
invalid([1:6, size(invalid,1)-5:size(invalid,1)],:)=1;
invalid(:,[1:6, size(invalid,2)-5:size(invalid,2)])=1;
% now we can see why nan was the better bad data marker
invalid=invalid | isnan(t4b) | isnan(t41b);
% we compute the standard deviation only for valid data
dt4b=std(t4b(~invalid));
dt41b=std(t41b(~invalid));

% next is the performing of the different fire tests
disp(' testing');
day=1;
```

```
night=2;
% the next line has to be replaced by the actual
% day/night detection
time=day;
% the threshold values are based on atbd-mod-14
threshold=[330 25 360 ; 315 10 330];
% now the five tests
firetest1=(band21>t4b+dt4b*4);
firetest2=band21>threshold(time,1);
firetest3=(t41>t41b+dt41b*4);
firetest4=t41>threshold(time,2);
firetest5=band21>threshold(time,3);
disp(' finding');
% combining of the tests
fire=(((firetest1 | firetest2) & (firetest3 | firetest4)) ...
      | firetest5) & ~invalid;
% fires can only be on land, with no clouds, and no snow
firepos=find(fire & land & ~snow & ~cloud);

% the actual fire detection is now done.
% rest of the script deals with map construction
disp([' number of fire pixels detected: ' num2str(size(firepos,1))]);
% testing is done, so we can clean up a little bit
clear firetestA firetestB firetesta firetestb firetestc;
clear t41 t41b t4b dt4b dt41b;
clear band2 band6 band10 band11 band12 band21 band31;
clear fire cloud snow baddata invalid land landseamask ndsi;

% now produce output image
disp('creating rgb image');
rgb=stretch(cat(3, band1, band4, band3));

% the following lines were used for debugging.
% If they are turned on again, they produce a nice
% image too.
%
% imagesc(rgb);
% rgb2=rgb;
% rgb2(firepos)=1;
% rgb2(firepos+size(rgb2,1)*size(rgb2,2))=0;
% rgb2(firepos+size(rgb2,1)*size(rgb2,2)*2)=0;
% rgb2(snowpos)=0;
% rgb2(snowpos+size(rgb2,1)*size(rgb2,2))=0;
% rgb2(snowpos+size(rgb2,1)*size(rgb2,2)*2)=1;
% rgb2(cloudpos)=0;
% rgb2(cloudpos+size(rgb2,1)*size(rgb2,2))=0;
% rgb2(cloudpos+size(rgb2,1)*size(rgb2,2)*2)=0;
% figure
% imagesc(rgb2)
%
% continue with the real stuff

disp('producing rectangular map');
disp(' calculating vectors and copying data');

% we don't want to loose data, so in the middle of the
% map a pixel should be 1km x 1km.
% one degree latitude should be 111.2km everywhere
% size of one degree longitude is dependent on latitude
% we use australia/adelaide/mawson lakes
ourlatitude=-34.86;
% the '0-' in the next line is there because latitudes are
% counted from south to north while matlab images from top
% to bottom
[nlat,latfac]=cstretch(0-latitude,111.2);
disp([' one degree latitude is ' num2str(latfac) ' pixels high']);
% cos(latitude) is the shrinking factor for longitude values
% this assumes a spherical earth which is perfectly right for
% this purpose
[nlong,longfac]=cstretch(longitude,111.2 * cos(ourlatitude/180*pi));
disp([' one degree longitude is ' num2str(longfac) ' pixels wide']);
% matlab indices start with 1
nlat=nlat+1;
nlong=nlong+1;
% create the target matrix to speed up copying
recmap=zeros(max(nlat(:)),max(nlong(:)),3);

% the next lines copy the data from rgb(:,:,:) to
% recmap(:,:,:) according to the information
% in from latitude/longitude
vector=sub2ind([size(recmap,1) size(recmap,2)], nlat(:), nlong(:));
recmap(vector)=rgb(:,:,1);
recmap(vector+size(recmap,1)*size(recmap,2))=rgb(:,:,2);
recmap(vector+size(recmap,1)*size(recmap,2)*2)=rgb(:,:,3);

% because not all pixels are of the same size, in recmap()
% are black spots. We want to fill them.
% this operation will need some memory, so clean up again
clear vector
clear cloudpos rgb snowpos
clear band1 band3 band4
pack;

disp(' smoothing (1/2)');
recmap=rgbfilter(recmap);
disp(' smoothing (2/2)');
recmap=rgbfilter(recmap);

%
% the code to paint a line grid may be inserted here
%

% next thing is to mark the fires with big red pixels
disp(' painting fires');
nfp=cat(2,nlat(firepos),nlong(firepos));
for i=1:size(nfp,1)
    recmap(nfp(i,1)+1,nfp(i,2)+1,:)=[1 0 0];
end

% if you want to see the result...
%
%figure;
%imagesc(recmap);
%

disp('writing image');
% imwrite tends to use a lot of memory
% so clear up and convert the data beforehand
clear band1 band3 band4 rgb rgb2 cloudpos ;
recmap=uint8(round(recmap*255));
pack;
imwrite(recmap,[filename '.fires.tiff'],'tiff');

% here we are!
```

```
finishtime=now;

% last things is to write the report
% if other information are needed in the report
% it is easy to modify it
disp('writing ASCII ouput');
msg=['# Results of firedetection' 10 ...
    '# input files:' 10 ...
    '#      ' filename '.geo.hdf' 10 ...
    '#      ' filename '.1000m.hdf' 10 ...
    '# output files:' 10 ...
    '#      ' filename '.fires.tiff' 10 ...
    '#           image range: ' num2str(min(latitude(:))) '/' ...
                                num2str(min(longitude(:))) ...
                   ' to '  num2str(max(latitude(:))) '/' ...
                           num2str(max(longitude(:))) 10 ...
    '#           pixelhigh:  ' num2str(1/latfac) ...
                 ' degrees latitude' 10 ...
    '#           pixelwidht: ' num2str(1/longfac) ...
                 ' degrees longitude' 10 ...
    '#      ' filename '.fires.txt (this file)' 10 ...
    '# starting time: ' datestr(starttime) 10 ...
    '# finishing time: ' datestr(finishtime) 10 ...
    '# ' 10 ...
    '# the following fires were found:' 10 ...
    '# latitude  longitude    posit-x    posit-y' 10 ...
];
% fopen() should not fail. If it fails, we can't do anything anyway
fid=fopen([filename '.fires.txt'],'w');
fwrite(fid,msg,'char');
fireinfo=cat(2,latitude(firepos),longitude(firepos), ...
                       nlat(firepos),nlong(firepos));
if isempty(fireinfo)
    fwrite(fid,'#NONE','char');
else
    fireinfo=sortrows(fireinfo,[3 4 1 2]);
    fprintf(fid,'%10.5f %10.5f %10.0f %10.0f\n', fireinfo');
end
fclose(fid);
% if we reach this point we are happy!
disp('all done');
return;


% -----
% some helpful internal functions
%

% stretching color in the rgb image

function res=stretch(inp);
mi=min(inp(inp~=0));
tmp=inp-mi;
ma= .3 * max(tmp(:));
res=tmp/ma;
res(res>1)=1;
res(res<0)=0;
return;

% stretching lat/long values to matrix coordinates

function [res, factor]=cstretch(inp,s);
```

```
secondfactor=1;
tmp=inp-min(inp(:));
factor=s*secondfactor;
tmp=round(tmp*factor);
tmp(isnan(tmp))=1;
res=tmp;
return;

% the filtering out of the black pixels

function recmap=rgbfilter(recmap);
fs=1; %size of the filter
xy=size(recmap);
disp('  finding black spots');
% first we need to find which pixels can be filled
recmapcopy=sum(recmap,3);
recmapcopy=conv2(recmapcopy,ones(2*fs+1),'same');
% exclude the edges again %% necessary??
recmapcopy([1:fs+1 xy(1)-fs-1:xy(1)],:)=0;
recmapcopy(:,[1:fs+1 xy(2)-fs-1:xy(2)])=0;
cxy=find(sum(recmap,3)==0 & recmapcopy~=0);
% all coordinates in cxy can potentially filled
% with data
disp('  count pixels we have got');
% how many nonzero neighbors has a pixel?
% we need this as a divisor in the averaging
divi=zeros(size(recmapcopy));
clear recmapcopy
a1=(2*fs+1);
% in the next line the '& 1' converts the double array
% in a logical array of ones and zeros.
% we then sum over these ones.
divi=conv2((recmap(:,:,1) & 1), ones(a1), 'same');
disp('  filtering');
disp('   red');
tmp=conv2(recmap(:,:,1),ones(a1),'same');
recmap(cxy)=tmp(cxy) ./ divi(cxy);
disp('   green');
tmp=conv2(recmap(:,:,2),ones(a1),'same');
recmap(cxy+xy(1)*xy(2))=tmp(cxy) ./ divi(cxy);
disp('   blue');
tmp=conv2(recmap(:,:,3),ones(a1),'same');
recmap(cxy+xy(1)*xy(2)*2)=tmp(cxy)./ divi(cxy);

return;
```

C.2 radiance2teff.m

```
function teff=radiance2teff(radiance,lambda);

% RADIANCE2TEFF convert radiance values to brightness temparatures
%
% input values:
%  radiance in W/m^2/um/sterrad as found in IMAPP HDF files
%  lambda in um
%  radiance can be a matrix
% ouput values:
%  teff effective brightness temperature in kelvin

% change history
%  08/11/2001 written by kendy kutzner from scratch using citation below
%  ??/11/2001 added computation of c1 and c2
%  ??/11/2001 documented


% citing http://ltpwww.gsfc.nasa.gov/MAS/masdug.html
% MODIS Airborne Simulator Research and Documents
%  Data User Guide
%  by Liam Gumley, Paul Hubanks, and Ed Masuoka  April 1994

% Conversion from IR radiance to Planck equivalent temperature
% or 'brightness temperature' may be done by inverting the Planck
% equation. The
% inverse equation is of the form
%
%   T(L,B) = C2 / L * loge ( C1 / ( L5 * B(L,T) * 106) + 1 )
%
% where,
%
% T(L,B) = brightness temperature in degrees Kelvin,
% C2 = ( h . c ) / k = 1.4387686 . 10-2 m K
% C1 = 2 . h . c2 = 1.1910439 . 10-16 W m-2
% l = wavelength in meters
% B(L,T) = Planck radiance in W m-2 sr-1 um-1

%Planck constant
h=6.62606876e-34;
%speed of light in vacuum
c=299792458;
%Boltzmann constant
k=1.3806503e-23;

%c1=1.1910439e-16;
%c2=1.4387686e-2;

%first radiation constant
c1=2*h*c^2;
%second radiation constant
c2=h*c/k;

radiance=radiance.*1e6;
%bring radiance to W/m^2/m/sterrad
lambda=lambda./1e6;
%bring wavelength to m
teff=c2./lambda./log(c1./(lambda^5 .* radiance)+1);
%voila!
```

C.3 attribute_read.m

```
function [res,f]=attribute_read(filename, sdsname, attrname);

%ATTRIBUTE_READ reads in Scientific Data Sets from .HDF files
%
% [res,status] = sds_read(filename, sdsname, attrname)
%
% filename is the name of the .HDF file to open. It is not parsed in
%  any way, so it may contain path information and it must contain
%  the extension .HDF
% sdsname is the name of the data set to be opened. It must appear exactly
%  as in the .HDF file.
% attrname is the name of the attribute to read. It must appear
%  exactly as in the .HDF file.
% res result. It may have more than one dimension and may not
%  be of type double.
% status contains 0 after successful execution, ~0 otherwise.

% change history
% 22/11/2001 written by kendy kutzner

f=0; % no failures until now

% open the file
sd_id = hdfsd('start', filename, 'read');
if sd_id==-1    % failure
    disp(['error: could not open file "', filename, '"']) ;
    f=-1;
    return ;
end

% search for data set and open it
sds_idx = hdfsd('nametoindex', sd_id, sdsname);
sds_id = hdfsd('select', sd_id, sds_idx);
if sds_id==-1 %failure
    outstring=sprintf('error: sds "%s" not found\n',sdsname);
    disp(outstring);
    f=-1;
    return;
end

%search for attribute and read it
attr_id=hdfsd('findattr', sds_id, attrname);
[res, status]=hdfsd('readattr', sds_id, attr_id);
if status~0
    disp(['Attribute "' attrname '" not found']);
    f=-1;
    return;
end

%close data set and file
stat1 = hdfsd('endaccess',sds_id);
stat2 = hdfsd('end',sd_id);
if (stat1~=0) | (stat2~=0)
    disp(['warning: problems closing file ' filename]);
end

return
```

C.4 sds_read.m

```
function [mat,f]=sds_read(filename, sdsname);

%SDS_READ reads in Scientific Data Sets from .HDF files
%
% [res,status] = sds_read(filename, sdsname)
%
% filename is the name of the .HDF file to open. It is not parsed in
%  any way, so it may contain path information and it must contain
%  the extension .HDF
% sdsname is the name of the data set to read. It must appear exactly
%  as in the .HDF file.
% res all the data. It may have more than two dimension and may not
%  be of type double.
% status contains 0 after successful execution, ~0 otherwise.

% written by kendy kutzner
% change history
% 25/10/2001 written from scratch
% 08/11/2001 added file closure


f=0;
mat=[];

sd_id = hdfsd('start', filename, 'read');
if sd_id==-1    % failure
    disp(['error: could not open file "', filename, '"']) ;
    f=-1;
    return ;
end

sds_idx = hdfsd('nametoindex', sd_id, sdsname);
sds_id = hdfsd('select', sd_id, sds_idx);
if sds_id==-1 %failure
    outstring=sprintf('error: sds "%s" not found\n',sdsname);
    disp(outstring);
    f=-1;
    return;
end
[dsname, dsndims, dsdims, dstype, dsatts, stat] = hdfsd('getinfo',sds_id);
if (stat~=0)
    disp(['warning: problems getting info for ' sdsname]);
end

ds_start = zeros(1,dsndims); % Creates the vector [0 0] where we want to start
ds_stride = []; % we don't skip anything
ds_edges = dsdims;  % and read to the end
[mat, status] = hdfsd('readdata',sds_id,ds_start,ds_stride,ds_edges);
if status~=0
    disp(['warning: problems reading sds ' sdsname]);
end

%close hdf file
stat1 = hdfsd('endaccess',sds_id);
stat2 = hdfsd('end',sd_id);
if (stat1~=0) | (stat2~=0)
    disp(['warning: problems reading sds ' sdsname]);
end

return
```

D Additional Matlab source files

D.1 `all500m2tiff.m`

With the growing number of available data files there were increasing difficulties to keep the overview which file contain which overpass. To solve these problems the small function `all500m2tiff.m` was written. The function converts all 500m HDF level1b files in `directory` to TIFF files by reading bands red, green and blue in 500m resolution, stretching contrast in each of them individually and combining them to a RGB image.

The only input parameter is the name of the directory containing the HDF files. The parameter must have a trailing '/'. The directory where the TIFF files are written is hardwired in the code of the function.

The function runs autonomously if no errors occur. A disadvantage is the need of memory, about twice the size of the largest HDF file.

```
function all500m2tiff(directory);

% ALL500M2TIFF(directory)
%
%   Converts all 500m HDF level1b files in directory
%   to TIFF files by reading bands red, green and blue
%   in 500m resolution, stretching contrast in each of
%   them individually and combining them to a RGB
%   image.
%
%   Caution: needs lot of memory (roughly twice the size of
%            the largest HDF file in directory)
%
%  directory: name of the directory containing the
%   HDF files. The parameter must have trailing '/'.
%   The directory where the TIFF files are written
%   is hardwired in the code of the function.
%
%  functions needed:
%   IMPORTHDF
%   STRETCH

% change history
%  25/10/2001 written by kendy kutzner from scratch
%  19/11/2001 documented by kendy


targetdir='/home/ingest/kendy/tiffs/';

f=dir([directory '*500m.hdf']);
s=size(f);
format compact;
for i=1:s(1)
    disp(['working on ' f(i).name ' (size: ' num2str(round(f(i).bytes/1024/1024)) 'M)']);
    [a,b,c,d]=fileparts(f(i).name);
    disp('reading..');
    [nir,red,green,blue]=importhdf([directory f(i).name]);
    clear nir;
    disp('stretching..');
    rgb(:,:,1)=uint8(round(stretch(double(red))*255));
    rgb(:,:,2)=uint8(round(stretch(double(green))*255));
    rgb(:,:,3)=uint8(round(stretch(double(blue))*255));
    clear red green blue;
    hdfml('closeall')
    whos rgb;
    disp('packing..');
    pack;
    disp('writing tiff..');
    imwrite(rgb,[targetdir b '.tiff'], 'tiff');
    disp('clearing up..');
    clear rgb;
    pack;
end;

function res=stretch(inp);
mi=min(inp(:));
tmp=inp-mi;
ma= .4 * max(tmp(:));
res=tmp/ma;
res(res>1)=1;
return;
```

D.2 hdf2bin.m

The DIMPLE software (see section A) can't read HDF files itself. As a way to export data from IMAPP to DIMPLE the Matlab function `hdf2bin` was written. It exports all data bands in 1km resolution to a binary file and some meta information from the geolocation process to another. These files can be read by Dimple via its raw files import facility.

To limit the resulting files in size, an area of interest can be provided and `hdf2bin` can cut the image along latitude and/or longitude values.

This script does no data processing in any way, it just exports the level 1b data to the binary file. Especially no corrections for the Bow Tie effect or pixel compression are done.

```
function f=hdf2bin(filename, latmin, latmax, longmin, longmax)

% status = HDF2BIN(filename, latmin, latmax, longmin, longmax)
%
%   Reads in HDF 1km & geo files, stores output in binary files
%   to be read by dimple or similar programs
%
%   filename is the basename of both HDF files. The extensions
%     .1km.hdf and .geo.hdf are added.
%   all remaining parameters give the boundaries where the data
%    should be cutted. If no cutting is desired, give
%    -90, 90, -180, 180 as values
%   status equal 0 after successful execution, -1 after an
%    error occured.
%
%   The filename of the output files are created with the
%    basename, then the resolution of the image after cutting
%    in pixels and the extension data.bin for the actual data
%    and lalohe.bin for the meta information latitude, longitude
%    and height.
% created 02/10/2001 by kendy kutzner
% modified 03/10/2001 by kendy
%   - now output are two files, one with single precision (lat, long & height)
%      the other with uint16 (data bands)
%   - bands are now in numerical order
%      (1 2 3 4 5 6 7 8 9 10 11 12 13lo 13hi 14lo 14hi 15 16 17
%       18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 )
%   - filename now with resolution in pixels
% modified 19/11/2001 by kendy
%   - added documentation

% some constants
nanreplace=0; % should invalid data be replaced by NaN?
rigidcut=1; % leave data outside specified region or not?


f=0; % no failure until now

%open the geo-file
actname=[filename, '.geo.hdf'];
sd_id = hdfsd('start', actname, 'read');
if sd_id==-1    % failure
    disp(['could not open file "', actname, '"']) ;
    f=-1;
    return ;
end

%reading in latitude
ds_data=readdsbyname(sd_id, 'Latitude', 2, nanreplace);
lat=ds_data;

%reading in longitude
ds_data=readdsbyname(sd_id, 'Longitude', 2, nanreplace);
long=ds_data;

%reading in heigh information
ds_data=readdsbyname(sd_id, 'Height', 2, nanreplace);
height=ds_data;

%finished with the .geo file
hdfml('closeall');

%open the 1000m-file
actname=[filename, '.1000m.hdf'];
sd_id = hdfsd('start', actname, 'read');
if sd_id==-1    % failure
    disp(['could not open file "', actname, '"']) ;
    f=-1;
    return ;
end

% lat, long and height have the same dimension, so
% cutting these three (lat, long, height) together
[area, xmin, xmax, ymin, ymax]=coordcut(lat, long, latmin, latmax, longmin, longmax);
nlat=lat(ymin:ymax, xmin:xmax);
nlong=long(ymin:ymax, xmin:xmax);
height=height(ymin:ymax, xmin:xmax);

bands=single([]);
bands=cat(3,bands,nlat);
```

```
bands=cat(3,bands,nlong);
bands=cat(3,bands,height);


% read the data and cut it as early as possible
ds_data=readdsbyname(sd_id, 'EV_250_Aggr1km_RefSB', 3, nanreplace);
tmp=ds_data(ymin:ymax, xmin:xmax,:);
bands=cat(3,bands,tmp);
ds_data=readdsbyname(sd_id, 'EV_500_Aggr1km_RefSB', 3, nanreplace);
tmp=ds_data(ymin:ymax, xmin:xmax,:);
bands=cat(3,bands,tmp);
ds_data=readdsbyname(sd_id, 'EV_1KM_RefSB', 3, nanreplace);
tmp=ds_data(ymin:ymax, xmin:xmax,:);
bands=cat(3,bands,tmp);
ds_data=readdsbyname(sd_id, 'EV_1KM_Emissive', 3, nanreplace);
tmp=ds_data(ymin:ymax, xmin:xmax,:);
bands=cat(3,bands,tmp);

hdfml('closeall');

%clean up a little bit, maybe we need mem
clear tmp;
clear ds_data;

% cut rectangular or along lat/long ?
if (rigidcut)
areaf=area(ymin:ymax, xmin:xmax);
j=size(bands);
for i=1:j(3) ;
bands(:,:,i)=single(double(bands(:,:,i)) .* double(areaf));
end;
end;

% now we are ready to write the data:
bandssize=size(bands);
count=length(bands(:));
if (bandssize(3)~=41) disp('error: not able to read 41 bands of data'); f=-1; return; end;
actname=[filename, '.', num2str(bandssize(1)), 'x', num2str(bandssize(2)), '.lalohe.bin'];
fid=fopen(actname, 'w');
if fid==-1
   disp('warning: could not open output file "', actname, '"');
   f=-1;
   return;
end;
%write lat, long, and height in 32 bit float precision
count=count-fwrite(fid,bands(:,:,1:3), 'single');
fclose(fid);
actname=[filename, '.', num2str(bandssize(1)), 'x', num2str(bandssize(2)), '.data.bin'];
fid=fopen(actname, 'w');
if fid==-1 disp('warning: could not open output file "', actname, '"'); f=-1; return; end;
%all succesive writes in uint16 precision
%250m bands 1, 2
count=count-fwrite(fid,bands(:,:,4:5), 'uint16');
%500m bands 3 4 5 6 7
count=count-fwrite(fid,bands(:,:,6:10), 'uint16');
%1km refl bands 8,9,10,11,12,13lo,13hi,14lo,14hi,15,16,17,18,19
count=count-fwrite(fid,bands(:,:,11:24), 'uint16');
%1km emissive bands 20,21,22,23,24,25
count=count-fwrite(fid,bands(:,:,26:31), 'uint16');
%1km refl band 26
count=count-fwrite(fid,bands(:,:,25), 'uint16');
%1km emissive bands 27,28,29,30,31,32,33,34,35,36
```

```
count=count-fwrite(fid,bands(:,:,32:41), 'uint16');
fclose(fid);
if count~=0 f=-1;
  disp(['warning: not all data written correctly (', ...
    num2str(count),' elements left)']);
end;

return;

%--------------------------------------------------------------------------
% helpful internal functions

%--------------------------------------------------------------------------
%getting sds_id by name
function sds_id=getsdsidbyname(sd_id, sdsname)

sds_idx = hdfsd('nametoindex', sd_id, sdsname);
sds_id = hdfsd('select', sd_id, sds_idx);
if sds_id==-1 %failure
    outstring=sprintf('warning: sds "%s" not found\n',sdsname);
    disp(outstring);
    f=-1;
    return;
end
return;

%--------------------------------------------------------------------------
%read all data from a given sd-set
function ds_data=readallsdsdata(sds_id, dsndims, dsdims)
ds_start = zeros(1,dsndims); % Creates the vector [0 0]
ds_stride = [];
ds_edges = dsdims;
[ds_data, status] = hdfsd('readdata',sds_id,ds_start,ds_stride,ds_edges);
return;

%--------------------------------------------------------------------------
%compute the cutting matrix
function [area, xmin, xmax, ymin, ymax]= ...
         coordcut(lat, long, latmin, latmax, longmin, longmax)

if latmin>=latmax disp('warning: latmin>=latmax'); end
if longmin>=longmax disp('warning: longmit>=longmax'); end
lat(lat<latmin | lat > latmax)=0;
lat(lat~=0)=1;
long(long<longmin | long > longmax) = 0;
long(long~=0)=1;

%area contains only 0 or 1, so
%convert area to uint8 to save memory
area=uint8(double(lat) .* double(long));
yx=size(area);
if length(yx)~=2 disp('dimension error'); return; end

%now look for boundaries
xmin=yx(2);
ymin=yx(1);
xmax=0;
ymax=0;
for y=1:yx(1)
    if max(area(y,:))>0
        ymin=min(ymin,y);
        ymax=max(ymax,y);
    end
end
for x=1:yx(2)
    if max(area(:,x))>0
        xmin=min(xmin,x);
        xmax=max(xmax,x);
    end
end
return;

%---------------------------------------------------------------------------
%read an entire data set by name
function ds_data=readdsbyname(sd_id, dsname, dimension, nanreplace)

sds_id=getsdsidbyname(sd_id, dsname);
if sds_id==-1 f=-1; return; end
[dsname, dsndims, dsdims, dstype, dsatts, stat] = hdfsd('getinfo',sds_id);
if (dsndims ~= dimension) % safety check
    disp(['warning: unexpected format in "' dsname '"']);
    f=-1;
    return;
end
ds_data=readallsdsdata(sds_id, dsndims, dsdims);
attr_idx = hdfsd('findattr',sds_id,'_FillValue');
[attr, status] = hdfsd('readattr', sds_id, attr_idx);
if status==-1
    attr=-999; % attribute not found, using default
    disp('Attribute "_FillValue" not found in "Latitude"');
end
if (nanreplace) ds_data(ds_data==attr)=nan; end %replacing fill values with NaN
return;
```

E Other source codes

This section lists other used source codes. This includes the programs FrameSync and slicer. Section E.3 describes the shell script which calls all other programs.

E.1 FrameSync.c

The entire `FrameSync.c` file would be far too long so only the differences to the original one are shown below. The difference file was created by the standard GNU diff program. For explanation of the changes please refer to section 3.3.

```
58d57
< int f_DumpFrameDecoded = FALSE; /* dump frames found after removing PN-randomization */
60d58
< int fframed = 1;
85,87d82
< fs_short Start_Sync_Match2[16];
<
<
126,130d120
< /* by kendy */
< int offbyonebyte = 0;
<
< int f_flywheel = FALSE;
< int nObits = 0;
160,161d149
<   puts
<   (" -4 [filename] = dump frame to the output file after removing PN-randomization");
283,307d270
<
<      case '4':
<        f_DumpFrameDecoded = TRUE;
<        if (option_type[i + 1] == OPT_TYPE_OTHER)
<  {
<    ++i;
< #ifdef _WIN32
<    if ((fframed =
<         open (option[i], O_RDWR | O_CREAT | O_TRUNC | O_BINARY,
<       _S_IREAD | _S_IWRITE)) < 0)
< #else
<    if ((fframed =
<         open (option[i], O_RDWR | O_CREAT | O_TRUNC,
<       0644)) < 0)
< #endif
<      {
<        sprintf (tbuf, "Couldn't open output file %s",
<         option[i]);
<        (void) MyError (1, tbuf);
<      }
<  }
<        else
<  (void) MyError (1, "-4 option requires a filename");
<        break;
<
```

```
513,514c476
< /*  if (bitshift > 7) */
<   if (bitshift > 15)
---
>   if (bitshift > 7)
549,550c511
<   /* for (i = 0; i < 8; ++i) */
<   for (i = 0; i < 16; ++i)
---
>   for (i = 0; i < 8; ++i)
555,573d515
< /**** version 2 of makeSyncSearchTable  ********/
< /* by kendy kutzner */
< /*  I'm working on bits 16-31 of sync pattern
<     because these contain valid bits for any possible bit shift
<     The values are stored in Start_Sync_Match2 */
<
<
< void
< Make_Sync_Search_Table2 (long pattern)
< {
<   int i;
<
<   for (i = 0; i < 16; i++)
<     {
<       Start_Sync_Match2[i] = pattern >> i;
<     }
< }
<
<
606,607c548
< /*  for (i = 0; i < 8; ++i) */
<   for (i = 0; i < 16; ++i)
---
>   for (i = 0; i < 8; ++i)
646,727d586
< /* Find_Start_Sync2 written by Kendy Kutzner */
< /*   much faster on noise data, but not as accurate as the orginal */
< /*   this doesn't matter because another check is done afterwards */
< int
< Find_Start_Sync2 (fs_long startsrch)
< {
<   int i, j, r;
<
<   for (i = startsrch; i < FrameSize; i++)
<     {
<       r = CVCDU_Raw_Frame[i + 1];
<       for (j = 0; j < 16; j++)
<  {
<    if (r == Start_Sync_Match2[j])
<      {
<        offset = i;
<        bitshift = j;
<        return (1);
<      }
<  }
<     }
<   return (0);
< }
<
<
<
```

```
749a609,610
>   static int f_flywheel = FALSE;
>   static int nObits = 0;
833c694
<       if (!Find_Start_Sync2 (startsync))
---
>       if (!Find_Start_Sync (startsync))
904,911c765
<       /*by kendy */
<       else
<  {
<    nObits = 0; /* sync failed, so reset consecutive good counter.
<     the slight chance to find a bad sync marker in the date
<     we accept */
<  }
<     } /*end for */
---
>     }
914d767
<   nObits = 0;
1362d1214
<   Make_Sync_Search_Table2 (ShortSync[0] * 0x10000 + ShortSync[1]);
1474c1326
<    if (VCDU_Primary_Header.Virtual_Channel_ID ==
<             Ch_ID_MODIS && VCDU_Primary_Header.SpaceCraft_ID == 0x2a)
<             /* 0x2a == TERRA */
---
>    if (VCDU_Primary_Header.Virtual_Channel_ID == Ch_ID_MODIS)
1485,1502d1336
<       /*****************************/
<       /***** If Dumping  decoded Frames *****/
<       /*****************************/
<        if (f_DumpFrameDecoded)
<  {
<    write (fframed, (char *) CVCDU_Raw_Frame, FrameSize);
<  /***** setup the next frame to check ******/
<    CVCDU_Raw_Frame[0] = lastword;
<    if (!ReadData ((char *) &CVCDU_Raw_Frame[1], FrameSize))
<      break;
<    lastword = CVCDU_Raw_Frame[ShortFrameSize];
<    continue;
<  }
< #endif
<
<
<
< #ifdef DEBUG
1561,1574d1394
<        /* added by kendy kutzner */
<        if (End_Of_Prev_Packet + Start_Of_Packet >
<    Max_Packet_Size)
<  {
< #ifdef DEBUG
<    fprintf (stderr,
<     "bogus telemetry packet size (nblks=%i, nFramesRead=%li\n",
<     nblks, nFramesRead);
< #endif
<    Have_Start_Of_Packet = FALSE;
<    continue;
<  }
<
<
```

```
1616c1436
<       if (ImageInfo.fout && f_flywheel)
---
>       if (ImageInfo.fout)
1728,1730d1547
<    /* inserted by kendy */
<    Start_Of_Packet += (Packet_Size + 7);
<    /*end insert */
1737c1554
<       if (ImageInfo.fout && f_flywheel)
---
>       if (ImageInfo.fout)
```

E.2 slicer.c

The program slicer is explained in section 3.1, based on `ctest.c` written by Ricky Luppino and shown below.

```
#include <errno.h>
#include <string.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdarg.h>

#define CHUNK 1024*1024
#define BLK 1024
#define FNS 300

int main(int argc, char *argv[])
{
    unsigned char buf[CHUNK];
//    unsigned char res[BLK];
    char filename[FNS + 1];
    char targetfile[FNS + 1];
    char tmpfile[FNS + 1];
    FILE *fp;
    FILE *of;
    int fileopen;
    int start_chunk;
    int chunk_count;
    int correl[BLK];
    int winval[BLK];
    int i, j, offset;
    int k;
    int count_array[256];
    int highest;
    int highest_v;
    int highcor;

    if (argc < 2) {
fprintf(stderr, "Usage \n slicer <filename> [<targetfile>] \n");
exit(1);
    }
    strncpy(filename, argv[1], FNS);
    if (argc > 2) {
strncpy(targetfile, argv[2], FNS);
    } else
strncpy(targetfile, filename, FNS);
```

```
    strncat(filename, ".raw", FNS);

    fp = fopen(filename, "rb");
    if (fp == NULL) {
fprintf(stderr, "could not open %s\n", filename);
exit(1);
    }
    snprintf(tmpfile, FNS, "%s.slicer%i.tmp", targetfile, getpid());
    chunk_count = 0;
    start_chunk = 0;
    fileopen = 0;

    // loop around and do chunks until EOF
    while (fread(buf, 1, CHUNK, fp) == CHUNK) {
for (offset = 0; offset < BLK; offset++) {
    j = offset;
    for (k = 0; k < 256; k++)
count_array[k] = 0;
    while (j < CHUNK) {
count_array[buf[j]]++;
j += BLK;
    }
    // find highest
    highest = 0;
    highest_v = count_array[0];
    for (k = 1; k < 256; k++)
if (count_array[k] > highest_v) {
    highest_v = count_array[k];
    highest = k;
}
    correl[offset] = highest_v;
    winval[offset] = highest;
}

// find max correl value in the CHUNK and print
highcor = 0;
for (i = 0; i < BLK; i++) {
    if (correl[i] > highcor)
highcor = correl[i];
}
chunk_count++;
// printf("chunk %d = %d\n", chunk_count, highcor);
fprintf(stderr,".");
if (highcor == BLK) {
    if (fileopen) {
fwrite(buf, 1, CHUNK, of);
    } else {
of = fopen(tmpfile, "w");
if (of == NULL) {
    fprintf(stderr, "could not open temp file %s\n",
    tmpfile);
    exit(1);
}
fileopen = 1;
fwrite(buf, 1, CHUNK, of);
start_chunk = chunk_count;
    }

} else {
    if (fileopen) {
// close file
fileopen = 0;
fclose(of);
snprintf(filename, FNS, "%s-%i-%i.raw", targetfile,
 start_chunk, chunk_count - 1);
// printf("attempt to rename %s to %s \n", tmpfile, filename);
printf("%s-%i-%i\n",targetfile, start_chunk, chunk_count -1);
if (0 != rename(tmpfile, filename))
    fprintf(stderr, "error: %s\n", strerror(errno));
    }
}
    }
    if (fileopen) {
fclose(of);
snprintf(filename, FNS, "%s-%i-%i.raw", targetfile,
 start_chunk, chunk_count);
// printf("attempt to rename %s to %s \n", tmpfile, filename);

printf("%s-%i-%i\n",targetfile, start_chunk, chunk_count );
if (0 != rename(tmpfile, filename))
    fprintf(stderr, "error: %s\n", strerror(errno));
    }
    fclose(fp);
    fprintf(stderr,"\n");
    return 0;
}
```

E.3 The shell script

This small shell script ties it all together. It calls the programs

- Slicer
- reformat
- FrameSync
- IMAPP consisting of
  - L0_to_L1a
  - geolocate
  - calibrate
- Matlab with the function `firedetection`

with all their necessary arguments. It also can update some files needed by IMAPP. The only parameter it takes is the name of the (freshly recorded) raw file.

```
#!/bin/bash
IMAPPDIR=/dd2/IMAPP_RUN2
#IMAPPDIR=/dd1/IMAPP_RUN
IMAPP=$IMAPPDIR/geoimapp.csh
SLICER=/home/ingest/bin/slicer
FS2=/home/ingest/bin/fs2
```

```
WGET=/usr/bin/wget
MATLAB=/app/matlab12/bin/matlab
REFORMAT=/home/ingest/bin/reformat
CP=/bin/cp
MV=/bin/mv

# we should check write permission in the current directory

# maybe the next four lines should appear in
# a weekly cronjob

#$WGET ftp://acdisx.gsfc.nasa.gov/pub/.dbs/ancillary/leapsec.dat
#$WGET ftp://acdisx.gsfc.nasa.gov/pub/.dbs/ancillary/utcpole.dat
$CP leapsec.dat /data/IMAPP/level1a/static
$CP utcpole.dat /data/IMAPP/level1a/static

# check for command line arguments


# start operation
$SLICER $1 > $1.slicer
for actfile in `cat $1.slicer` ; do
  echo "working on file $actfile";
  # turn the bytes arround
  # this would not be neccesary with fs2 but with stps
  # so do it
  ls $actfile* -l
  $REFORMAT < $actfile.raw > $actfile.tmp
  ls $actfile* -l
  $MV $actfile.tmp $actfile.raw
  # IMAPP calls the level0 files .PDS,
  # in ITR they are called .CCSDS
  #
  # in some future the FS2 line may be replaced with a line
  # calling STPS
  $FS2 -i $actfile.raw -o $actfile.ccsds ;
  $IMAPP $actfile.ccsds ;
  # check the results of IMAPP
  #
  echo firedetection\(\'$IMAPPDIR/$actfile\'\) | $MATLAB -nojvm -nosplash;
done
```

F Configuration Files

This section list necessary configuration files, most of them for the STPS suite of programs.

F.1 socapps.cfg

```
F2S nugget 8003 32000 0 1 0 1
EDOS_F2S nugget 8003 10000 0 1 0 1
S2F stps28 1999 32000 0
S2S stps28 5000 600 edos1 5001 600 0 1
*END*
```

F.2 stps.cfg

```
VERSION 3.41
CHANNEL 0

CC_MODE 1
CC_ENGR_DISPLAY 1
DISPLAY_STRING MY--TelemetryFromFileToFile

ERROR_LOG_FILE_DIR ./my
STATUS_LOG_FILE_DIR ./my

DSP_PROCESS_UP 0
DSP_DEVICE 0
DSP_SOCKET_SERVER_PORT 8002
DSP_SOCKET_OUTPUT_ON 1
DSP_SOCKET_SEND_DATA_HEADER 1
DSP_SESSION_TIME_OUT 30
DSP_RECORD 0
DSP_REC_DIR_NAME ./my

INPUT_BUFFER_SIZE 655360
INPUT_DATA_TYPE 0

CHECK_INPUT_BLOCK_CRC 0

REC_DATA_MAX_FILE_SZ -1
REC_DATA_TIME_LIMIT -1

DLTP_UP 1
DLTP_FRAME_DATA_TO_FILE 0
DLTP_PROCESSED_DATA_DIR /home/kendy/stps/my
DLTP_SOCKET_HOST_MACHINE ingest
DLTP_SOCKET_CLIENT_PORT 8003
```

```
DLTP_SOCKET_RECV_DATA_HEADER 0

FS_frameLength 1024
FS_bitReverseFrames 0
FS_lossOfLockBits 0
FS_autoPolarity 0
FS_pnDecodeCCSDS 0
FS_pnDecodeBeforeReverse 0
FS_noFlywheelFramesFlag 0
FS_outputFramesState 2
FS_framesPerMode0 3
FS_framesPerMode1 9
FS_framesPerMode2 3
FS_framesPerMode3 1
FS_bitFlipsPerMode0 3
FS_bitFlipsPerMode1 3
FS_bitFlipsPerMode2 0
FS_bitFlipsPerMode3 3
FS_bitSlipsPerMode0 3
FS_bitSlipsPerMode1 3
FS_bitSlipsPerMode2 0
FS_bitSlipsPerMode3 3
FS_fsPatternLength 4
FSP0 1A
FSP1 CF
FSP2 FC
FSP3 1D

CHECK_FRAME_CRC 0

RSD_ENABLED 1
RSD_BITS_PER_SYMBOL 8
RSD_MAX_CORRECTABLE_ERRS 16
RSD_Mo 112
RSD_POA 11
RSD_VIRTUAL_FILL 0
RSD_INTERLEAVE 4
RSD_MODE 1

__EXTERNAL_MODULES_INPUT_PARAMS_BEGIN__
_EDOS_BEGIN_

ESH_VER 1
TGT_PORT AA

RSDheaderOnly 0
EDOSSocOutPort 2005
EDOSoutputServerHost nugget
EDOSdiscardBadFrames 1
```

```
EDOSuseUNIXtime 0
EDOSinFreq 1000000
EDOSoutputSync 1
EDOSoutputCLCW 0
EDOCclcwPortNum 0
EDOSclcwMulticastAddress 225.2.7.000
TERRAspecificPNDECODE 1

__EXTERNAL_MODULES_INPUT_PARAMS_END__

_END_PARAMS_
```

F.3 pk.cfg

```
# Configuration File Format
#  all comments must have a "#" in the first column
#  all parameters must have a space in the first column
#
# Total frame length including sync, and RS symbols if present
TOTAL_FRAME_LENGTH= 1024

# RS interleave, set to 0 if RS symbols are not present
RS_INTERLEAVE= 4

# Base directory where packet files are placed
PKT_DIRECTORY= .

#For each VC to process, enter parameters, no spaces or
# comments are allowed in between parameters.
# 1= yes, 0 = no

VCID= 1

INSERTZONE_SIZE= 0
CLCW_PRESENT= 0
CRC_PRESENT= 0
```

(VC 2-40 have been cut out to shorten this document.)

```
VCID= 41
INSERTZONE_SIZE= 0
CLCW_PRESENT= 0
CRC_PRESENT= 0

VCID= 42
INSERTZONE_SIZE= 0
CLCW_PRESENT= 0
CRC_PRESENT= 0
```

```
VCID= 43
INSERTZONE_SIZE= 0
CLCW_PRESENT= 0
CRC_PRESENT= 0
```

(VC 44–62 have been cut out to shorten this document.)

```
VCID= 63
INSERTZONE_SIZE= 0
CLCW_PRESENT= 0
CRC_PRESENT= 0
```

References

[1] Steve Ackerman, Kathleen Strabala, Paul Menzel, Richard Frey, Chris Moeller, Liam Gumley, Bryan Baum, Crystal Schaaf, and George Riggs. *Algorithm Theoretical Basis Document Disriminating Clear-Sky from Cloud with MODIS.* MODIS Cloud Mask Team, November 1997.

[2] Ronald E. Alley and Marit Jentoft-Nilsen. *Algorithm Theoretical Basis Document for Brightness Temperature.* Jet Propulsion Laboratory, April 1999.

[3] Stuart Ellis AM. *1999-2000 Annual Report of the South Australian Country Fire Service.* South Australian Country Fire Service, June 2000.

[4] M.D. Andrews and P.E. Ardanuy. A generalized scenario for cloud detection using modis-n. In *Geoscience and Remote Sensing Symposium, 1990. IGARSS '90*, pages 1487–1489, 1990.

[5] L.L. Bourgeau-Chavez, P.A. Harrell, E.S. Kasischke, and N.H.F. French. The detection and interpretation of alaskan fire-disturbed boreal forest ecosystems using ers-1 sar imagery. In *Geoscience and Remote Sensing Symposium, 1995. IGARSS '95*, volume 2, pages 1246–1248, 1995.

[6] S.A. Christopher, Min Wang, K. Barbieri, R.M. Welch, and Shi-Keng Yang. Satellite remote sensing of fires, smoke and regional radiative energy budgets. In *Geoscience and Remote Sensing, 1997. IGARSS '97*, pages 1923–1925 vol.4, 1997.

[7] South Australian Tourism Commission. *About South Australia*, 2001. `http://www.southaustralia.com.sg/mapway7.htm`.

[8] Consultative Committee for Space Data Systems. *Spacecraft ID List*, 2001. `http://nssdca.gsfc.nasa.gov/anon_dir/active/iacg/ccsds/ccsds.id`.

[9] Department of Land Administration. *Satelite Remote Sensing Services - Firewatch - Hotspot Detection*, December 2001. `http://www.rss.dola.wa.gov.au/newsite/noaafd/NOAAfd.html`.

[10] Dundee Satellite Recieving Station Home Page. *Dundee Satellite Receiving Station MODIS downlink packet format.* `http://www.sat.dundee.ac.uk/modisformat.html`.

[11] CRC for Satellite Systems. *ASTRA*. Institute for Telecommunication Research, 2001. `http://www.itr.unisa.edu.au/crcss/astra/astra.html`.

[12] Liam Gumley, Paul Hubanks, and Ed Masuoka. *MODIS Airborne Simulator Level-1B Data User Guide*, April 1994. `http://ltpwww.gsfc.nasa.gov/MAS/masdug.html`.

[13] Dorothy K. Hall, Andrew B. Tait, George A. Riggs, and Vincent V. Salomonson. *Algorithm Theoretical Basis Document for the MODIS Snow-, Lake Ice- and Sea Ice Mapping Algorithms*, October 1998.

[14] Jun ichi Kudoh. Forest fire detection in far east region of russia with noaa-15 in 1998. In *Geoscience and Remote Sensing Symposium, 1999. IGARSS '99 Proceedings*, volume 1, pages 182–184, 1999.

[15] Yoram Kaufman and Chris Justice. *Algorithm Theoretical Basis Document MODIS FIRE PRODUCT*. MODIS Science Team, 1998.

[16] K. Kawano, J. Kudoh, and S. Makino. Forest fire detection in far east region of russia by using noaa avhrr images. In *Geoscience and Remote Sensing Symposium, 1999. IGARSS '99 Proceedings*, volume 2, pages 858–860, 1999.

[17] A.G. Klein, D.K. Hall, and G.A. Riggs. Improving the modis global snow-mapping algorithm. In *1997 IEEE International , Volume: 2*, pages 619–621 vol.2, 1997. IGARSS '97. Remote Sensing - A Scientific Vision for Sustainable Development.

[18] Zhanqing Li, A. Khananian, R.H. Fraser, and J. Cihlar. Automatic detection of fire smoke using artificial neural networks and threshold approaches applied to avhrr imagery. *Geoscience and Remote Sensing, IEEE Transactions on*, 39(9):1859–1870, September 2001.

[19] Lockheed Martin Missiles & Space. *Interface Control Document (ICD) Data Format Control Book for EOS-AM Spacecraft (ICD-106)*. IS20008658C.

[20] Lockheed Martin Missiles & Space. *Direct Access System User's Guide for EOS-AM Spacecraft (ICD-107)*, November 1998. IS20008696.

[21] J. Minardi, G.B. Marchisio, and R.P. Treder. Spatial linear modeling and forecasting of forest fires across the united states. In *Geoscience and Remote Sensing Symposium, 1999. IGARSS '99*, pages 290–292 vol.1, 1999.

[22] MODIS Characterization Support Team (MCST). *Modis Geolocation Version 2 Product Format*, March 2001. `ftp://modis-xl.nascom.nasa.gov/pub/stig_temp/mlinda/www/LatestFilespecs%/MOD03.geolocation.fs.txt`.

[23] NASA. *Aqua Project Science*, 2001. `http://aqua.nasa.gov`.

[24] NASA. *NASA's Visible Earth*, 2001. `http://visibleearth.nasa.gov/Sensors/Terra/MODIS.html`.

[25] National Center for Supercomputer Applications at the University of Illinois. *HDF User's Guide Version 4.1r4*, December 2000. `ftp://ftp.ncsa.uiuc.edu/HDF/HDF/Documentation/HDF4.1r4/Users_Guide`.

[26] Mash Nishihama, Robert Wolfe, David Solomon, Frederick Patt, Jeffrey Blanchette, Albert Fleig, and Edward Masuoka. *MODIS Level 1A Earth Location: Algorithm Theoretical Basis Document Version 3.0.* MODIS Science Data Support Team, August 1997.

[27] Arno Peters. *Die neue Kartographie / The new cartography.* Friendship Press, 1983.

[28] Robert E. Wolfe, David P. Roy, and Eric Vermote. Modis land data storage, gridding, and compositing methodology: Level 2 grid. *Geoscience and Remote Sensing, IEEE Transactions on , Volume: 36 Issue: 4*, 36(4):1324–1339, July 1998.

[29] Yilin Zhao. *Vehicle location and navigation systems.* Artech House, 1997. Appendix B.