Norsk Regnesentral
NORWEGIAN COMPUTING CENTER

Note

SnowLab

A System for Automated Snow Product Generation

Software documentation

Note no SAMBA/20/06

Authors Jostein Amlien
Hans Koren
Line Eikvil
Rune Solberg

Date 12 December 2006

Norsk Regnesentral

Norsk Regnesentral (Norwegian Computing Center, NR) is a private, independent, non-profit foundation established in 1952. NR carries out contract research and development projects in the areas of information and communication technology and applied statistical modeling. The clients are a broad range of industrial, commercial and public service organizations in the national as well as the international market. Our scientific and technical capabilities are further developed in co-operation with The Research Council of Norway and key customers. The results of our projects may take the form of reports, software, prototypes, and short courses. A proof of the confidence and appreciation our clients have for us is given by the fact that most of our new contracts are signed with previous customers.

Norsk Regnesentral
Norwegian Computing Center
Postboks 114, Blindern
NO-0314 Oslo, Norway

Besøksadresse
Office address
Gaustadalléen 23
NO-0373 Oslo, Norway

Telefon · telephone
(+47) 22 85 25 00
Telefaks · telefax
(+47) 22 69 76 60

Internett · internet
www.nr.no
E-post · e-mail
nr@nr.no

Title	**SnowLab - Software documentation**
Authors	**Jostein Amlien** **Hans Koren** **Line Eikvil** **Rune Solberg**
Date	12 Dec 2006
Year	2006
Publication number	SAMBA/20/06

Abstract

This note is a documentation of the SnowLab software system for the automatic generation of snow related geophysical products from MODIS satellite data. The system is designed for the daily generation of Snow Covered Area (SCA), Snow Temperature Surface (STS), Snow Grain Size (SGS), and Snow Surface Wetness (SSW). In addition, the cloud cover is generated for masking purposes.

The generated products can be categorized as basic products or as derived products. The basic products are generated from one satellite pass only. The derived products are generated from more than one pass, usually based on the basic products. The derived products part also includes multi-sensor functionality. The basic products are SCA, STS and SGS. The derived products are SSW and an improved SCA.

The software system is designed as a production chain, implemented in IDL/ENVI, and runs on a Linux platform. It takes care of the whole process from downloading of MODIS data to the final products are generated.

Keywords	Snow products, geophysical parameters, SCA, STS, SGS, SSW
Target group	Snow hydrologists, hydropower companies
Availability	Open
Project number	236 065
Research field	Remote sensing
Number of pages	41

Contents

1 System overview

1.1 Purpose

This report describes and documents a software system for the automated retrieval of various snow products from remote sensing data. The basic idea behind the software system is to consider the production process as a production chain, consisting of certain steps, as in this generic production chain:

- Data download
- Data import
- Pre-processing
- Product generation
- Product export

The production chain is controlled by a production chain framework, which retrieves data and leads them through the various steps in the production chain.

The actual steps in the chain are defined by the application software. Although the current application is the generation of snow products, the application software could in principle address virtually any production purpose.

The software for the generation of snow products is organized as modules that are closely related to the steps in the production chain.

This document describes the production chain and its individual steps. It documents the various software modules, and how they interact with other modules and with the framework.

1.2 System architecture

This section introduces the main modules in the system, the main process of the production chain, and the main principles for the framework that controls the dataflow through the system.

1.2.1 Modules

As shown in Fig. 1 the application software is grouped into five main modules. Each module provides at least one function that interacts with the framework through a standardized interface.

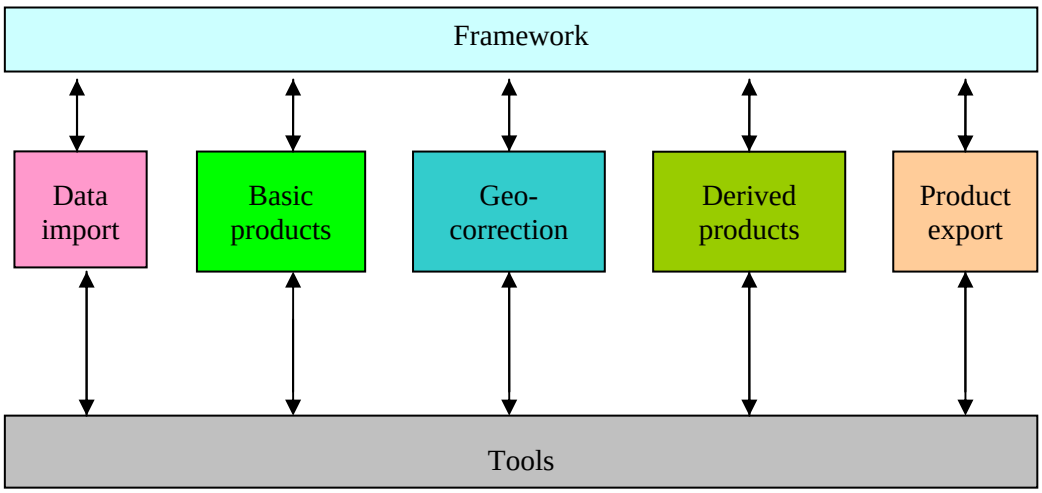


***Figure 1** : The main modules*

- *Data import module*: Provides functions for converting downloaded satellite image data to ENVI format (the internal format in the software system)
- *Basic products module*: Provides snow parameter retrieval functions that require one input scene only. The module also ensures that the basic products are generated in the correct sequence
- *Geo-correction module*: Provides functions for establishing the geometrical reference of a satellite scene, as well as the resampling of the basic products to that reference
- *Derived products module*: Provides functions that combine a multi-temporal set of basic products into a derived snow product
- *Product export module*: Converts the products into user specified format
- *Toolbox*: Provides basic tool functions to the other modules

1.2.2 Main process

The main process for the retrieval of cryospheric products are shown in Fig 2. The arrows show the sequence of the various functional steps. The colours in the diagram refer to the main modules above. When derived products are not requested, the process proceeds directly to the export step.

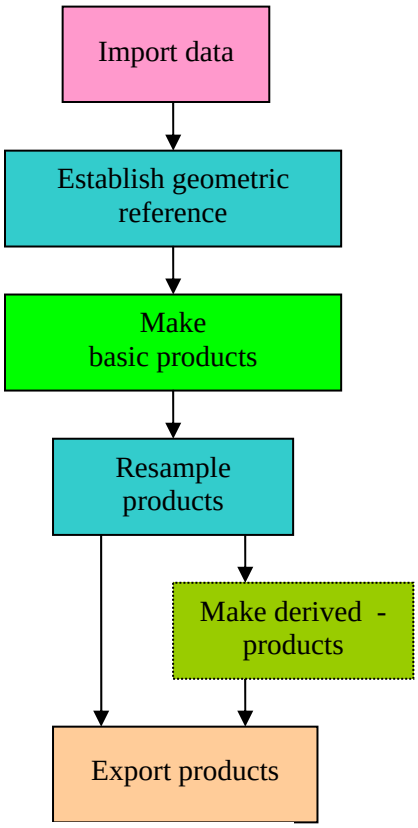


Figure 2. *The main process*

Two functions in the geo-correction module are called in separate steps. The establishment of the geometrical reference of the scene should be undertaken before the generation of basic products, since this step may require some map oriented auxiliary data. The resampling of the basic products into map geometry builds on the geometrical reference and is run once for each basic product.

1.2.3 The framework

The framework is intended for automatic processing of remote sensing datasets through a production chain. A simple controller controls the production chain where one dataset is processed at the time, running the process through the necessary steps.

The main principle behind the interaction between the application software and the framework is that the framework does not need to know the application software. The framework can thus be applied for any application. This principle is obtained by:

- Defining a standardized API (Application Program Interface)
- Saving all processing results in files between each step
- Transferring application specific arguments through text files

No functions performing operations on the data are part of the framework, but should all be given in the application specific modules. Any application can be plugged into the framework, if it is possible to call its functions from the framework's method API, which is described more detailed in section 3.2.

The framework takes care of retrieving the input data from some source and making them available to the application software. It controls the sequence in which the application functions are being called, and the dataflow through the system.

The framework is written in IDL and C++, and is intended to run on a Linux platform. Fig. 3 gives an overview of the framework.

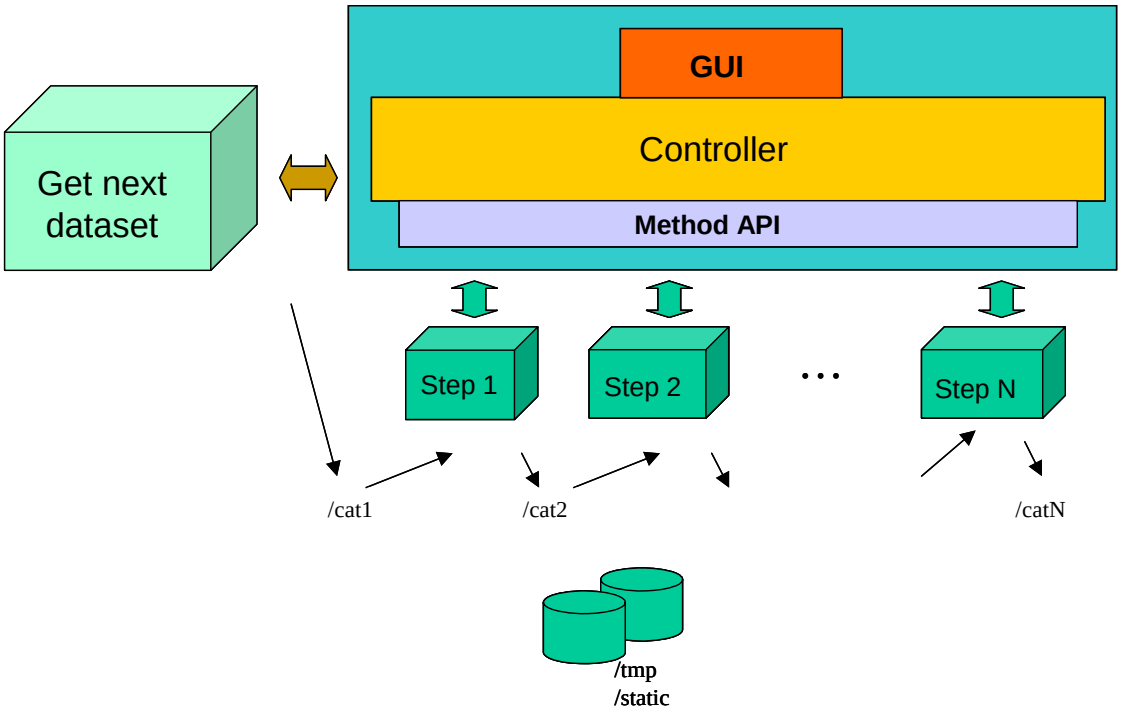


Figure 3. *The framework*

1.3 Overview of the system modules and their main functions

The system consists of the framework, the production modules, and a toolbox of common functions. This section gives an overview of the main functions in the production modules. These functions correspond to the main steps described above, and are callable from the

framework, following the requirements outlined in section 1.2.3. Also some functions at a lower level follow these requirements.

1.3.1 Import module

The import module extracts data from MODIS images and stores it as ENVI files. It comprises these main functions:

- Import MODIS file
- Retrieve MODIS spatial subset (optional)

1.3.2 Basic products module

This module controls the production of basic products, i.e. products that can be retrieved from one scene. It comprises these main functions:

- Cloud Mask module
- Snow Covered Area (SCA) module
- Surface Temperature of Snow (STS) module (optional)
- Snow Grain Size (SGS) module (optional)

1.3.3 Geometric correction module

The geometric module consists of two main functions, which may be called from separate parts of the production chain. These two functions are:

- Establish the geometric correspondence between image and map geometry.
- Resample basic products into map geometry

1.3.4 Derived products module

This module handles products where one needs to consider multiple basic products, e.g. in a time-series, or derived from multiple sensors. A central part of this module is a time-series / multi-product controller, which all main function relies on. These functions are:

- Snow surface wetness
- Snow distribution pattern
- SCA multi-scene/time-series combination
- SCA multi-sensor/time-series combination

1.3.5 Export module

The purpose of the export module is to export the product to a format tailored to the end-user of the product.

1.4 Outline of the production chain

The production chain is controlled by the framework. The main steps mainly correspond to the main functions in the modules.

- Import data
    a. Import MODIS data
    b. Retieve spatial subset (optional)
- Establish geometric reference
- Basic products generation
    a. Make cloud mask
    b. Make SCA – Snow Covered Areas
    c. Make STS – Snow Temperature Surface (optional)

    d. Make SGS – Snow Grain Size (optional)
- Resample to geometric reference
- Time-series 'derived ' products (optional)
    a. Make SSW (Snow Surface Wetness) : based on STS and recent change in SGS
    b. Make multi-scene SCA: estimate current SCA using a time-series of basic SCA
    c. Make multi-sensor SCA: estimate current SCA using multiple sensors
- Export products

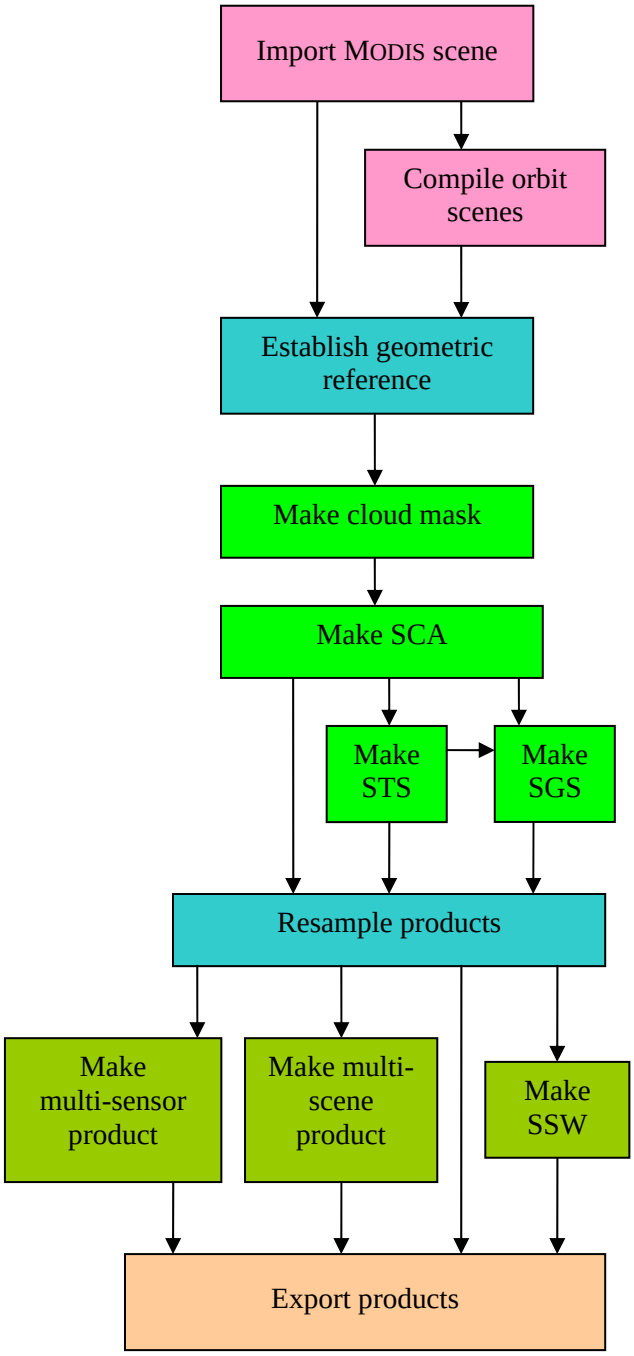


***Figure 4**. Outline of the production chain*

1.5 Snow parameter retrieval algorithms

The basic snow parameters that are produced are Snow Covered Area (SCA), Surface Temperature Snow (STS), Snow Grain Size(SGS), and Snow Surface Wetness (SSW).

1.5.1 Snow Covered Area (SCA)

The algorithm for the retrieval of SCA is known as the Norwegian Linear Reflectance-to-Snow-Cover (NLR) algorithm. The algorithm is based on an empirical reflectance-to-snow-cover model originally proposed for NOAA AVHRR by Andersen (1982) and later refined by Solberg and Andersen (1994). The algorithm has recently been tailored to MODIS data by NR. It retrieves the snow-cover fraction for each pixel. The model is calibrated by providing two points of a linear function relating observed reflectance or radiance to fractional snow-cover area (see Figure 5). The calibration is usually done automatically by means of calibration areas. Statistics from the calibration areas is then used to compute calibration points for the linear relationship.

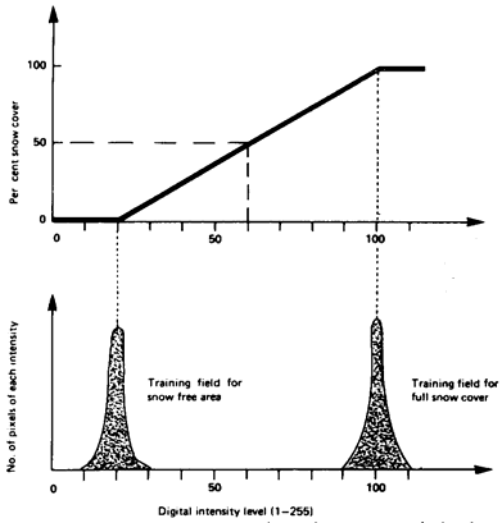


Figure 5 *The Norwegian Linear Reflectance-to-Snow-Cover (NLR) algorithm illustrated. A pixel value is linearly transformed to a snow cover percentage for that pixel. The algorithms are based on the assumption that the bare-ground reflectance is constant. (Andersen 1982)*

This algorithm and it s validation has been reported in more detail from the project Envisnow, see D1-WP3, Part 1 (Solberg et al. 2005).

1.5.2 Cloud Cover Mask

A particular problem for practical use of the snow algorithms has been cloud detection. NR has experimented with several approaches, and the current best cloud detection algorithm is based on K Nearest Neighbour (KNN) classification of MODIS data. In a KNN classifier a pixel, represented by a vector of band values, is assigned the label, which is most prevalent among the K nearest labelled vectors from a reference set. A KNN classifier is an asymptotically optimum classifier as the size of the reference set increases. This algorithm has been described in more detail in the Envisnow report D1-WP3, Part 1 (Solberg et al. 2005).

1.5.3 Surface Temperature Snow (STS)

The retrieval of STS is based on Key's algorithm (Key 1997), which has been calibrated for various AVHRR sensors as well as for Modis, see http://stratus.ssec.wisc.edu/products/surftemp/. The application of Key's algorithm for the retrieval of snow surface temperatures in Norwegian mountains has been examined by Amlien and Solberg (2004). The main idea is to correct for the atmospheric attenuation by a modified split-window technique, where the atmospheric path-length is taken into account.

1.5.4 Snow Grain Size (SGS)

The idea behind the SGS parameters is that the spectral signature of snow depends on the grain size. The spectral signature curves show a clear drop in reflectance when moving towards the longer wavelengths. The effect is more prominent for larger snow grains. The method has been described and evaluated by Koren et al (2004). The algorithm compares the reflectance in MODIS bands 2 and 7 and calculates a grain size index, defined as SGS = (M2-M7)/(M2+M7).

1.6 Derived product algorithms

The derived snow parameters that are produced are SnowCoveredArea multi-product (multi-SCA), and SnowSurfaceWetness (SSW). Common for the derived products are that they are derived from a set of basic products.

1.6.1 Snow Surface Wetness (SSW)

The main idea is to compare the temporal development in the SGS with the current value of STS. If the temperature is close to 0 °C and the grain size is increasing, it can be assumed that the snow is becoming wet.

The temperature observations give a good indication of where wet snow potentially may be present, but are in themselves not accurate enough to provide very strong evidence of wet snow. However, a strong indication of a wet snow surface is a rapid increase of the effective grain size observed simultaneously with a snow surface temperature of approximately 0°C. The algorithm can be expressed in a simplified version as

$$\begin{aligned}
&\text{if } (SGS_{\text{today}} - SGS_{\text{recently}} > \text{SGS}_{\text{snowmelt-tresh}}) \\
&\qquad \text{and } (STS_{\text{low}} < STS_{\text{today}} < STS_{\text{high}}) \text{ then} \\
&\quad SSW = \text{WET-SNOW} \\
&\text{else} \\
&\text{if } SGS_{\text{today}} < \text{SGS}_{\text{bare-ground-tresh}} \text{ then} \\
&\quad SSW = \text{SNOW-FREE} \\
&\text{else} \\
&\text{if } STS_{\text{today}} > STS_{\text{high}} \text{ then} \\
&\quad SSW = \text{SNOW-FREE} \\
&\text{else} \\
&\quad SSW = \text{DRY-SNOW}
\end{aligned} \tag{2}$$

Note that more temperature classes are used in the implemented algorithm. Also a threshold of the SCA product is applied in order to mask out snow-free areas. The algorithm can also infer bare ground from temperature observations above 0°C and a rapid developing negative gradient for SGS (both due to appearance of snow-free ground patches at the sub-pixel level).

1.6.2 Snow distribution pattern

The snow distribution pattern of a local area, like a drainage area, can be estimated from the SCA product and a snow distribution model for that local area. The snow distribution model is an empirical one, based on classifications of the snow cover in a series of high-resolution images, like Landsat. The series must be representative for the development of the snow cover during a typical melting season. The model is represented by a likelihood function that gives the sequence in which the pixels will become snow-free.

The algorithm for the snow distribution pattern goes like this: Retrieve the SCA for the local area from the SCA product. Then produce a corresponding SCA from the empirical model by applying a threshold of the likelihood function. This mask produced by this threshold is the estimated snow distribution pattern.

1.6.3 Multi-SCA

Due to cloud cover problems it is difficult to obtain an updated product every day. In order to overcome these problems, the SCA value must be predicted by means of recent results from the same sensor (multi-temporal), or from a different one (mult-sensor).

Each observation is given a confidence value, which are declining with the time. As long as the confidence value is above a threshold, the observation is considered as useful. When new data arrives, the SCA value is updated where the confidence for the new observations are better than the old ones. The confidence depends on the cloud cover and the view angle, in addition to the time since the acquisition.

The multi-SCA algorithms are described by Solberg et al (2004a, 2004b, 2005b), and Malnes et al (2005). This algorithm has also been described in a more detail in the Envisnow report D1-WP3, Part 1 (Solberg et al. 2005).

2 System Operator's Manual

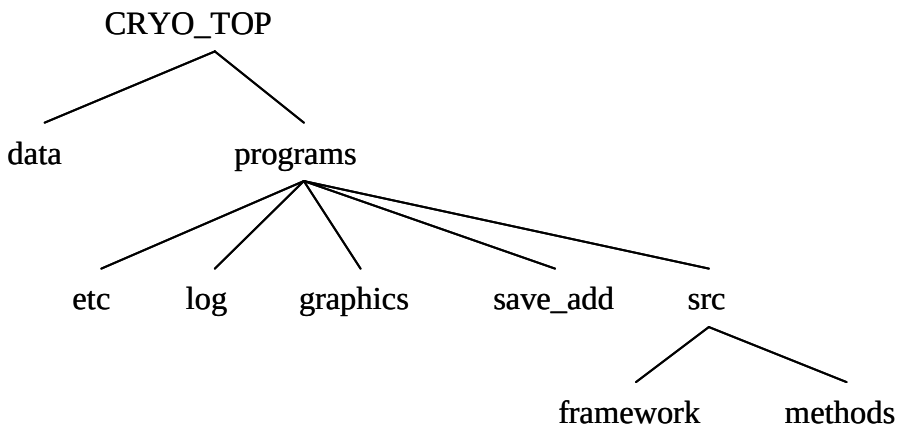
2.1 System Installation Guide

2.1.1 Unpacking the software

The software of the framework is contained in the tar-file cryo.tar. To install the framework, do the following:

- Copy the tar-file to the directory where you want to put the software.
- Unpack the tar-file:
  `> tar –xvf cryo.tar`
- Go to the sub directory cryo/programs:
  `> cd cryo/programs`
- Run the setup script cryo_setup from this catalogue:
  `> source cryo_setup`
  The environment variable CRYO_TOP will be set to the top directory where the software was unpacked, and IDL/ENVI variables and configurations will be set up.

2.1.2 Directory structure



The contents of these directories will be as follows:

data	Intended for various data. It is the default directory for image data, when no other is specified (see description of the contents of the image directory below). It is also the default directory for list files containing specifications of local datasets
programs	Main directory for the software. It also contains setup and configuration files.
etc	Contains ftp-configuration files for data providers.
log	Contains logfiles.
graphics	Contains logos etc.
save_add	Contains .sav files (compiled IDL code).
src	Contains source code.

framework Source code for the framework, if made available

methods Source code for the methods, if made available

The data directory will initially be empty, except that *static/* will contain various files needed for the application.

As the datasets are processed, sub-directories holding the intermediate results from each step of the processing chain, will be created under the image directory, and given names based on the data type, the current step in the process and the original name of the dataset:

```
<datatype>/out*/<dataset>
```

2.1.3 Setup of the production chain

Although the production chain is started from a GUI available in the ENVI menu, its behaviour is controlled by the user, but by the predefined configuration files. These files are being defined when a production chain is being set up, and it is not intended that these files could be changed by the system operator.
The behaviour of the system will be changed by the manipulation of these configuration files. This section presents the main principles for the configuration files and gives an overview of how they are used in the current system.

2.1.3.1 Configuration file format

The general format of the configuration file is simple. The configuration file is a text file that contains keyword-value pairs, where the keyword is separated from the value with an '=' sign: <keyword> = <value>. The value may be one single item or a list of items.

2.1.3.2 Main configuration file

The main configuration file is always named `current.cfg` and always located under the `programs/` directory (see above description of the directory structure). This file will be read once as the program is started.

The purpose of the main config file is to define:
- the parameters needed by the automatic downloading routine of the framework,
- one or more production chains
- a few optional parameters

Configuration of automatic download

The automatic download part is controlled by parameters defining one or more data providers, as well as a local mail user receiving messages about data ready to be downloaded.

The specification of a local mail user is required for receiving messages that will trigger the automatic remote download of data:

mailusr user name
mailpwd password
mailhost mail host, e.g. mail.nr.no
mailport local port e.g. 110

The various data providers are identified by:

nof providers number of data providers
providers list of data providers

<provider> *sender*: email-address that each of the providers will use to send the message

For each provider there is also a separate configuration file, <provider>.cfg, located in `programs/etc/`. The ftp-download from that provider is specified by these parameters:

host	name of ftp host, e.g. ftp.nr.no
user	user name, e.g. anonymous
pass	password, e.g. anonymous

Configuration of the production chains

There should be specified one chain for each datatype, e.g .one for MODIS and one for AVHRR.

The parameters controlling the production chains are:

nof datatypes:	number of datatypes
datatypes:	list of datatypes

For each datatype there should be defined a production chain

<datatype> *nof steps*	number of steps for this specific datatype
<datatype> *steps*	list of the steps in the production chain for a specific datatype

Each step is specified by a describing name, the function to call, an input dataset, an output dataset, and a configuration file

Optional main configuration parameters

The other main parameters are optional:

project	default is 'cryo'
maxLog	maximum size of logfile in Bytes, default is 100KB
image catalog	full path to an existing directory, default is $CRYO_TOP/data

2.1.3.3 Setup of the production steps

In the current system, there is defined a production chain for MODIS data. This is done through the definition of the parameter *modis steps* in the main configuration file. This parameter comprises several items that represent the steps in the production chain. The steps are performed in the sequence that they are listed in the *modis steps* parameter.

Each step is defined by a corresponding item in the parameter *modis steps*. The item will typically fill one line in the main configuration file, consisting of several strings that represent:
- a reference name (one or more strings) to be shown in the GUI
- name of function to call (one string)
- input directory (one integer)
- output directory (one integer)
- the name of a configuration file (one string)

The following example shows how the production chain could be defined in the main configuration file. Note that the parameter *modis nof steps* will determine how many steps in the chain that actually will be performed.

```
modis nof steps = 4          # Number of steps in the chain
modis steps = {
import               import_modis_data          1  2  importModis.cfg,
make products        make_products              2  3  makeProd.cfg,
resample products    resample_products          3  4  resampl.cfg,
export products      export_product             4  5  export.cfg   }
```

The system programmer may control what will happen in each step by editing the corresponding configuration files, according to section 2.3. As a general rule the operator should not change these files.

- The import step will consider a list of MODIS swath files and import specified subsets of these files

- The make basic products step will consider a list of basic products and make these products. Although the list of products to make is defined in the corresponding configuration file, *makeProd.cfg*, the sequence of their processing is controlled by the *make_products* step
  As an example, assume that the configuration file *makeProd.cfg*, contains the product definition list `products = {sca, geo, cloud }.`
  The products will not be generated in that sequence, but rather as {geo, cloud, sca}, since the makeSca function requires a geometric reference grid as well as a cloud mask in order to work.
  The basic products from this step must be considered as intermediate products only.
- The resample step will consider a list of basic products and resample them into a specified map projection. A geometrical reference grid has already been established in the previous step, and will be utilized.
- The export step will consider a list of products and export them to final products intended for some users.

2.2 System Operator's Guide

The CRYO production chain for snow products are implemented as a plug-in to the ENVI software. It is started from a simple GUI, but the interactions during the processing steps are kept to a minimum.

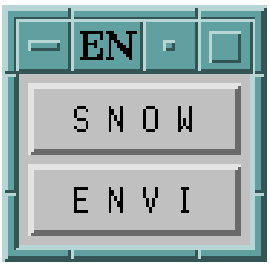
From the GUI, the user may choose to
- perform ENVI and IDL commands
- run the production chain in remote modus
- run the production chain in local modus
- stop the processing

2.2.1 Starting the software

The production chain should be run on a *linux* platform. In order to start the production chain, the user needs to run a setup program, and then start ENVI:

```
> setenv IDL_PATH '<IDL_DEFAULT>'
> source cryo_setup
> envi
```

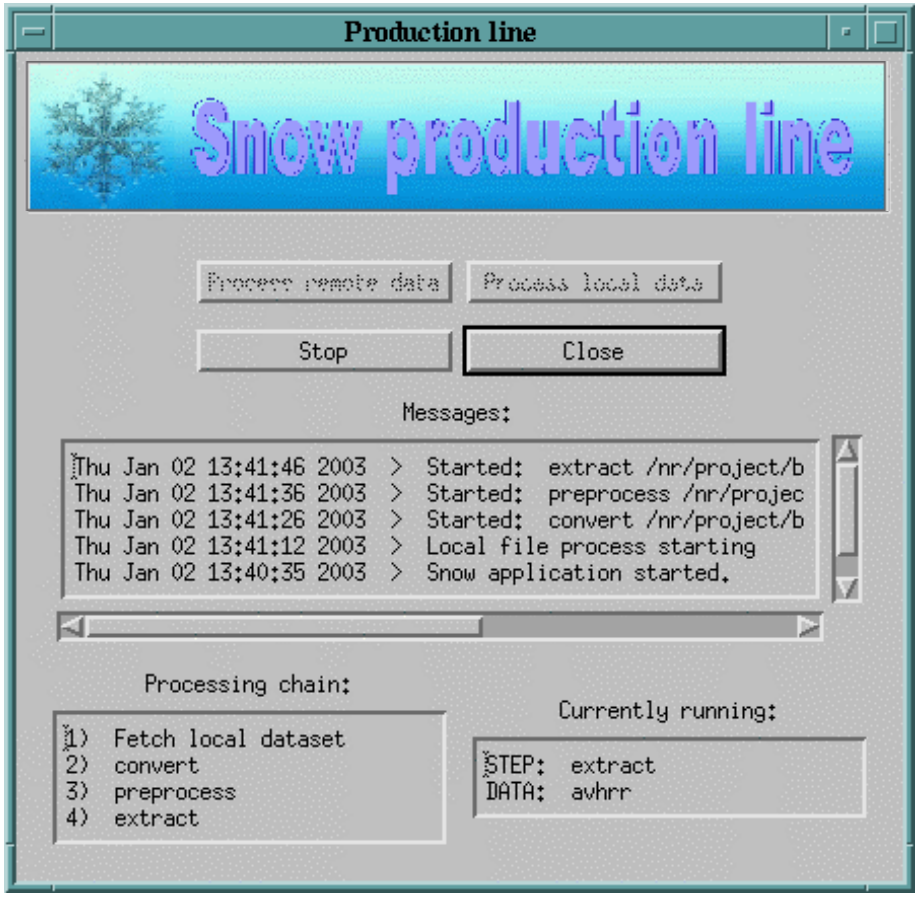
Now the user will see the ENVI prompt and may enter ENVI and IDL commands from that prompt. In addition, a simple menu will be available:



The lower 'ENVI' button gives access to the complete ENVI menu. By clicking the button labelled 'SNOW', the user will start the automatic production chain. Note that in some implementations, the label 'SNOW' have been substituted by 'CRYO'. After calling the automatic production chain, the GUI on the next page becomes available, and the prompt are being locked for input.

The user may choose to start the production chain in local modus ('process local data') or in remote modus ('process remote data'). In both modi the central field of the GUI will report back the progress through the various steps. Only the remote modus are utilizing the automatic downloading facilities in the framework.

The user may stop the processing by clicking the 'Stop' button. By clicking the 'Close' button, the control is returned to the ENVI prompt and the initial menu will be made available.



The operator may monitor the progress by means of the GUI. The lower left pane gives an overview of the steps in the chain. The lower right pane identified the step that is currently being processed. The central pane yields a more detailed log of the steps in the chain, including if some steps return with an error.

2.2.2 Remote modus

When run in remote modus, the system will wait for emails to arrive. Each email will be parsed and the specified files will be downloaded and put through the production chain according to the given data type. When a dataset has been processed, the system will wait for the next email to arrive. The control may be returned to the user by pushing the 'stop' button.

2.2.3 Local modus

The local modus of the production chain does not utilize the automatic downloading facilities in the framework, but requires that the datasets exist somewhere in the local file system.

By selecting the local modus, the user will get access to ENVI's file selector in order to select a text file containing a list of local datasets. A dataset is a set of files, representing a single scene, residing in a separate directory. The dataset list file will contain one line for each dataset. For each dataset the list will specify the full path to the directory. In addition, the data type is specified in order to identify the corresponding production chain.

When run in local modus, the system goes through the list in sequence and processes one dataset at the time undertaking all steps required for the given data type. When all datasets are processed, the control is returned to the production line menu.

A more detailed set of instructions for running the chain in local modus is given in section 2.3.

2.2.4 Process control

The user may stop the processing by pushing the 'stop' button. In some cases it may take some time for the system to react to the stop request.

Temporary files

Note that the directory *tmpdir* in the current system will not be strictly temporary. They may also contain files needed by the software for the management of multi-temporal data series. As a general rule assume that temporary files will be removed by the software itself.

Logfiles

Log messages from the system will be written to the command window, and to the logfile, which resides in the *programs/log* directory. When the logfile *cryo.log* reaches its maximum size it will be backed up as *cryo.log.prev*. If it is required to save more of the logging info, the user should make copies of the logfile regularly.

Error handling

If the system aborts, the ENVI software should be terminated and then restarted. However, a lot of error situations are managed by the software without aborting the processing. In these cases, the processing of the current dataset will be terminated and the processing of the next dataset will be started.

2.3 Instructions for operating in local modus

The steps to be handed by the operator when operating in local modus are:

- Download data
- Edit .lst file
- Start production chain
- Upload products

2.3.1 Data download

Create a directory for the new dataset

```
>     mkdir <path_to_modis_data>/yyyy.mm.dd_hhmm
```

Download data from KSAT by means of ftp. The dataset are represented by two products of level1b (MOD021KM and MOD02QKM). Download both of them.
The data should be found in the MODIS-testdata directory on the ftp-server ftp3.ss.no.

```
>     cd <path_to_modis_data>/yyyy.mm.dd_hhmm
>     lftp -u norskr,<passwd> ftp3.tss.no
lfpt  cd MODIS-testdata
lftp mget MOD021KM* MOD02QKM*
```

2.3.2 Edit list file

Edit the file *$SNOW_TOP/data/modis_sca.lst* (or make a new file *modis_yyymmdd.lst*)

For each scene to be processed, the file should contain one line like:

```
      <path_to_modis_data>/yyyy.mm.dd_hhmm    modis
```

2.3.3 Start the processing chain

```
>
>     setenv SNOW_TOP <path_to_processing_chain>
>     cd $SNOW_TOP/programs
>     setenv IDL_PATH '<IDL_DEFAULT>'
>     source cryo_setup
>     envi
```

From the GUI:

- Select 'SNOW' and then 'Snow process'
- Select 'Process local data'
- Pick the appropriate .lst file (the one you just edited)
  $SNOW_TOP/data/modis_sca.lst

2.3.4 Upload the product

The end product from the processing chain is located in the directory:
$SNOW_TOP/data/modis/out4/yyyy.mm.dd_hhmm

The .png file in this directory can easily be inspected visually. It should show the sea, clouds, and SCA values.

Copy the product directory to the outgoing FTP:

```
>     cp $SNOW_TOP/data/modis/out4/yyyy.mm.dd_hhmm <outgoing_ftp>
```

3 System Developer's Manual

3.1 Introduction

The application software is programmed in IDL, with calls to ENVI. Some functions are implemented in C++, and are available as system calls to executable binaries.

The application software is organized into software modules, providing one or more functions that can be called by the framework as a step in the production chain.

3.2 Interaction between the framework and the application software

Since the framework does not know the application software, the application functions must fulfil some requirements in order to interact with the framework (see sect. 1.3). This section describes these requirements.

The application program interface (API) in the framework will call the application functions through this specified function call:

```
status = funcName(  dataset_in, dataset_out, $
         staticdir, tmpdir, logfile, configFileName )
```

The return value is the error status, which may force the production chain to terminate. The arguments are all input arguments that specify catalogues and files that are made available to the application function.

The first two arguments are determined by the corresponding step in the main configuration file and by the dataset_name defined by the input data (the directory name of the input in local modus or by the triggering e-mail in remote modus)

```
dataset_in  = CRYO_TOP/data/modis/outx/<dataset_name>
dataset_out = CRYO_TOP/data/modis/outy/<dataset_name>
```

The following three arguments are invariant

```
staticdir   = CRYO_TOP/data/static
tmpdir      = CRYO_TOP/data/tmp
logfile     = CRYO_TOP/programs/log/cryo.log
```

The last argument is specified in the main configuration file

```
configFileName = makeProd.cfg
```

This can be illustrated by an example: Consider the second step in the production chain for modis data as specified in section 2.1.3.3. The item

"make products make_products 2 3 makeProd.cfg,"

will define the following function call:

```
status = make_products( CRYO_TOP/data/modis/out1/<dataset_name>, $
                        CRYO_TOP/data/modis/out2/<dataset_name>, $
                        CRYO_TOP/data/static,                    $
                        CRYO_TOP/data/tmp,                       $
                        CRYO_TOP/programs/log/cryo.log,          $
                        makeProd.cfg)
```

Note that out1 and out2 correspond to the numbers 2 and 3 respectively.

3.2.1 Main principles

The dataflow through the chain is controlled by the two first arguments 'dataset_in' and 'dataset_out', which refer to two different catalogues that are unique to each specific scene. Thus they are dynamic arguments that will depend on the actual scene being processed

The argument ‘dataset_in’ in one function will typically be identical to the argument ‘dataset_out’ in a function called in an earlier step in the production chain. This catalogue thus serves as the connection between the two steps.

The other arguments always refer to the same catalogues or files, independent of what scene that is being processed. These catalogues and files are thus common data.

Since the framework lack the possibility to provide the functions with application specific arguments, all such arguments are transferred by means of configuration files. The system operator controls the production chain by managing the configuration files.

- Input catalogue – *dataset_in*
  The input catalogue is dynamically set by the framework. It will typically be the output of a former step in the production chain.
- Output catalogue – *dataset_out*
  The output catalogue is dynamically set by the framework. It will typically become the input of a following step in the production chain.
- Static catalogue – *staticdir*
  The static catalogue is a fixed catalogue where static data should be found. Such data may be configuration files, water masks, training areas, class definitions, colour tables, etc. The static data may be organized into sub-directories.
- Temporary catalogue – *tmpdir*
  The temporary catalogue is a fixed catalogue where temporary file could be put by the application software. The files in this catalogue could be deleted at any time they are not being actively used.
- *Logfile*
  This argument identifies a fixed text file intended for appending log-messages.
- Configuration file – *configFileName.*
  This argument identifies the filename of a text file that contains the input parameters required by the application software. The application software described in this document assumes that this file is found in *staticdir*. The format of the configuration files is described in Appendix.
- Return value – *status*
  A successful completion should return a positive integer.

3.2.2 Modifications and specifications

This section describes the practices that have been followed concerning the file and catalogues referred in the API. Note that the practice concerning temporary files may be considered as a violation or modification of the main principles for the framework.

Temporary files

In order to manage multi-temporal datasets, some common available data catalogue had to be made available for temporary versions of a multi-temporal product. The problem is that the framework does not provide a catalogue for this purpose. The chosen strategy was to store such temporary multi-temporal products in the scene-specific catalogues and their references in the *tmpdir* catalogue. This catalogue should therefore not be deleted.

Since the role of the *tmpdir* catalogue was extended to manage multi-temporal scenes, all temporary files relating to specific scenes were put into the scene-specific catalogues *dataset_in* and *dataset_out*. Typically, temporary subdirectories are being used for this purpose in the current application software.

Input/output catalogues

These catalogues may contain temporary subdirectories, as described above.

The output catalogue may already contain data produced in a preceding step or sub-step, typically when more than one product are produced. In some cases the application functions may need to use these files as input files.

The input catalogue should never be used as an output catalogue, but the application software may delete temporary files that are no longer needed.

Configuration files

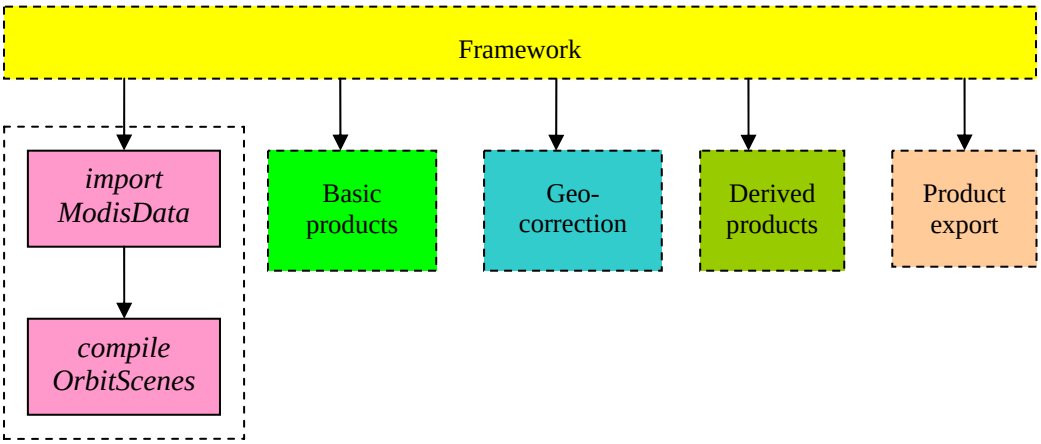
The *configFileName* parameter may contain any string variable. It is assumed that this string contains the name of a file residing in the *staticdir* catalogue, and that this file is a configuration file, i.e. a text file that follows the configuration file format defined in the framework.

The current software does not expect to find all configuration parameters directly in the file referred to by the *configFileName* parameter. Instead it is expected that this file refers to other configuration files, which may contain configuration parameters common to more than one function. The intension of this strategy is to reduce the risk of inconsistent configuration data.

3.3 Module descriptions

In this chapter each module is described in more detail. For each module its main functions are described. All the functions are named like *functionName*, while the modules will be referred to as 'module name' in this chapter.

3.3.1 Module data import



The data import module consists of one main function and an optional one:

- *importModisData* is the main function and reads MODIS images and stores specified image layers (spectral subsets) of them as ENVI files
- *compileOrbitScenes* is an optional function called from the *importModisData* function. It compiles scenes in the same orbit into one contiguous scene.

3.3.1.1 Function importModisData

The purpose of the function *importModisData* is to import subsets of MODIS datafiles, including radiometric data, geolocation data and view angle data.

Input data

MODIS calibrated data (MOD02) stored as *.hdf files. Depending on the format, there may also be an additional *.hdf.met-file or an xml-file for each *.hdf-file

The input catalogue may contain MODIS files of these types:

- MOD021KM: Image data, 1 km resolution
- MOD02HKM: Image data, 500 m resolution
- MOD02QKM: Image data, 250 m resolution

The scenes are in the original acquisition geometry (swath geometry).

The hdf-files contain a lot of data, including:

- Calibrated image data, represented as integers, and supplied with calibration coefficients for converting the integers to real radiance values. When applicable, there are also calibration coefficients for reflectance values.
- Geo-location info, for points regularly distributed over the image grid
- View angle info, for points regularly distributed over the image grid. Note that this data field is contained in MOD021KM only.

Output data

The output ENVI files are given filenames according to to this format:
<content>_<resolution>_swath_ modis _yyyyddmm_hhmm

For each specified input image, there may be generated three ENVI-files with different content:

data: the specified spectral bands from the hdf-file, stored as integers

latlon: the latitude and longitude for the geo-location points, stored as a two-layer float image with one cell for each geo-location point

angle; the view angle from the hdf-file, stored in a one-layer integer image. The angle image will be expanded to the same dimensions as the data image. Note that angle data will be produced for MOD021KM only.

In addition to the ENVI-files and the corresponding *.hdr-files, there will be generated a few other corresponding files:

*.*rad*: text-file containing calibration coefficients for converting 'data' to radiance,
If * refer to an 'angle' file, the rad-file should be interpreted as scaling coefficients for converting the integers to float values representing degrees

*.*ref*: text-file containing calibration coefficients for converting 'data' to reflectance,
set to zero for the non-reflexive spectral bands

*.*frm:* text-file containing meta data for 'data' and 'angle'

Configuration data

The configuration file may contain these keywords:

imgTypes: list of image types, allowed values are '1KM', '500', '250'
250_import: name of cfgFile, triggered if imgTypes contains '250'
500_import: name of cfgFile, triggered if imgTypes contains '500'
1KM_import: name of cfgFile, triggered if imgTypes contains '1KM'

Each of these files provides these configuration arguments for their respective imgType:
imgType: for identification only; must correspond to above
bandnames: list of band names identifying the spectral bands to retrieve
angleFields: list of angle 'bands' to read, should specify 'SensorZenith'
latLon: flag (0 or 1) whether latlon data should be read for this imgType

Interactions

Called from framework

Calling: *compileOrbitScenes* if requested, but this function is not required for KSAT data.

Optional function compileOrbitScenes

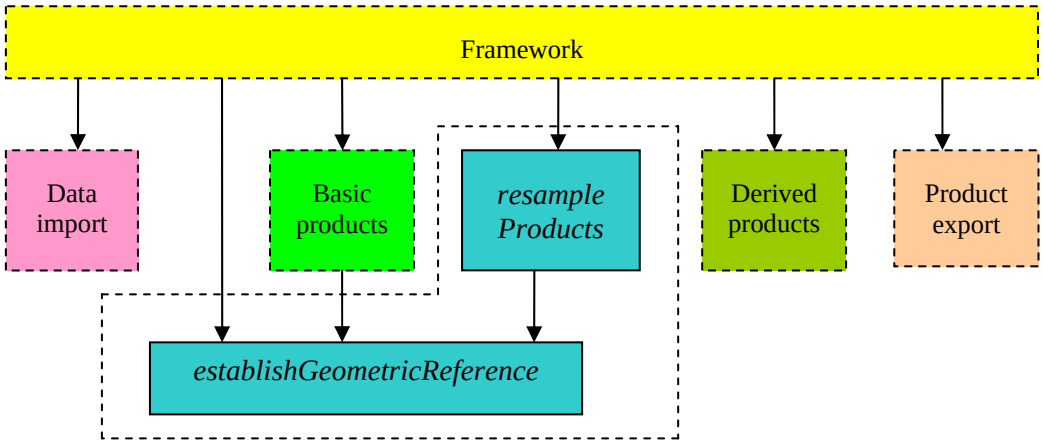
The purpose of the c*ompileOrbitScenes* function is to compile scenes from the same orbit into one contiguous scene that covers a given region of interest (ROI) from north to south.

The function is called from the *importModisData* function as an option. It will be trigged if *importModisData* identifies the configuration parameter *compileOrbitCfg*. Its value will be the name of cfgFile to be used by the *compileOrbitScenes* function.

This function substitutes the current imported scene with a compilation of the newly imported scenes that fulfil the requirements. Before completing the compiled orbit-scene redundant parts of it are removed, i.e. scans that are clearly outside the given ROI will be removed.

The configuration parameters for this function are omitted in this version of the document, because this part of the system is not required when using data from KSAT.

3.3.2 Module geometric correction



The geometric correction module provides two functions, both callable from the framework through the API:

- *establishGeometricReference* is a function that establishes the geometrical relation between a specific scene and some given geometrical reference.
- *resampleProducts* is the function that actually projects the specific image data into the given geometrical reference system.

The execution of *establishGeometricReference* is always required before executing *resampleProducts*. It may also be required to call it before some of the basic product functions. The function *establishGeometricReference* may therefore be called from the framework, from the relevant basic product functions, and from the *resampleProducts* function.

3.3.2.1 Function establishGeometricReference

The purpose of the *establishGeometricReference* function is to establish the geometrical relation between a specific scene and some given geometrical reference. The geometric correspondence will be stored as a geo-index map, which points from the map into the original image. Each cell

in this geo-index map points to a location in the input image. Correspondingly it also points into any product derived form that image.

Input data

The function requires an image and its corresponding *latLon* file.

Configuraton data

The configuration file defines the two parameters used for the identification of the image that the geometric reference should refer to. It also identifies a separate configuration file that identifies a grid in a specific map projection.

product: identifies product type of reference image, default = '*data'*
cell size: identifies imgType of reference image {250 | 500 | 1KM}
projCfg: name of configFile specifying the geometric reference

This configFile specifies the map projection and area by means of these parameters:
Projection: projection type, default 'UTM'
UtmZone: triggered if projection is UTM
Datum: datum name, default is 'WGS-84'
UpperLeft: map projection coordinates of upper left corner
LowerRight: map projection coordinates of lower right corner
projName: to be used for identification / reference

Output data

An *index* file is representing the specified geometrical reference between the specified map geometry and the image. Each cell in this geo index represents a cell in the map grid, and contains a position in the un-rectified image.

The index file is put into a sub-catalogue identified by the *projectionName.* In that way it is possible to recognise index files produced in earlier steps in the production chain. It should also be possible to produce and recognize several geometric reference grids, if ever required.

Interactions

Called from: framework, *makeBasicProducts*, *resampleProducts*

One call should be sufficient.

3.3.2.2 Function resampleProducts

The purpose of the *resampleProducts* is to take an image or a basic product and resample it into the map geometry according to the geo-index map that represents the established geometric reference.

Input data

The input data is a list of basic products from the *makeBasicProducts* module (or alternatively image data from the *dataImport* module), and the map reference grid produced by the *establishGeometricReference* function.

Configuration data

The resampling of the products is specidfied like this:

products: list of products to resample
**_resamplMet*: the method to use for a given product

**_conf*	flag if also the confidence file should be resampled
projCfg:	name of cfgFile for the projection definition
cell size	cell size of output products

Output data

For each of the specified products, one product file and optionally one confidence file are resampled to the specified map projection.

Interactions

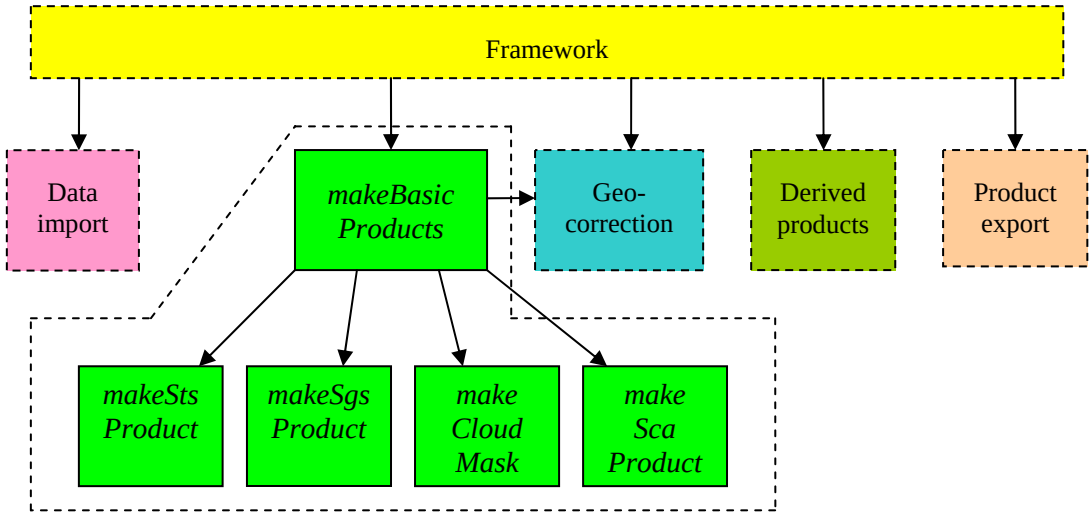
Called from framework

Calling e*stablishGeometricReference* if required

3.3.3 Module basic products

This module undertakes the production of basic products. By basic products are meant products that can be retrieved from one single scene. This input scene will typically be in image (swath) geometry. Although the input data in principle may have been re-sampled into some map (grid) geometry before making the basic products, this is generally not recommended and is therefore avoided in this system. Therefore the *latlon* file will always be copied forward in the processing chain.

The basic products that can be produced by this module are: Cloud Cover, Snow covered Area (SCA), Surface Temperature Snow (STS), and Snow Grain Size (SGS).



3.3.3.1 Function makeBasicProducts

This function serves as an organizer of the basic product generation functions. Essentially it produces all requested products from the data stage to the basic product stage. When necessary, it controls the sequence in which the products are generated. It also ensures that some common configuration parameters are consistent between the product generation functions.

This function makes sure that cloud cover is generated before the SCA, and that a proper geographic reference grid is made available to the SCA function. The geographic reference grid produced is available for later user in the production chain, like any other basic product.

Input data

The input directory contains ENVI files produced by the data import module.

data: files of the required types,
latlon: files (will be copied forward, unless data files are already resampled)
angle: data for 1KM data only, to be used in cloudCover and in STS

Configuration data

The configuration data comprise some common parameters, as well as a list of products to make.

The list of products is defined by specifying the cfgFiles that should be used in the functions that produces the requested products, e.g.

cloudCfg: cfgFile for CloudCover, will trigger the production of cloudCover
scaCfg: cfgFile for SCA, will trigger the production of SCA
stsCfg: cfgFile for STS, will trigger the production of STS
sgsCfg: cfgFile for SGS, will trigger the production of SGS
map_*.cfg cfgFile for geometric reference in resolution *,
will trigger the function *establishGeometricReference*

One important issue for the function *makeBasicProducts* is to process the cloud mask and make the geographical reference grids before the other products. Therefore, independent of the sorting of the cfgfile names above, the *makeBasicProducts* will ensure that the reference grids and the cloud cover will be produced before the snow products.

One parameter is not used directly in this function, but is passed forward to the specific functions that make the products.

The last parameter is a reference to a file that identifies the class codes used in the cloud cover product. It is passed forward to those product specific functions that require these class codes.
maskCodeFile: list class codes for the CloudCover product

Output data

product: files and their corresponding *confidence* files
latlon files, if present in input

Interactions

Called from framework

Calls a number of product specific functions when requested in the configuration. All these functions are described elsewhere in this document

stablishGeometricReference
makeCloudCover
makeSCA
makeSTS
makeSGS

3.3.3.2 Function makeCloudCover

The purpose of the *makeCloudCover* function is to make a cloud mask from the image data.

The cloud mask is the result of a kNN classification of the MODIS data. It requires 1KM data, but may produce a cloud mask in a different cell size.

The cloud cover function also produces a basic confidence product in 1KM cell size based on the cloud cover and the view angle.

Input data

The input directory contains ENVI files produced by the data import module.

data file	should represent a MODIS scene of type 1KM that includes the MODIS bands 1, 4, 6, 19, 20, 26, and 31
angle file:	representing the view angle, see function *inportmodisData*

Configuration data

maskCodeFile:	specifies the classCodes to be used in the output product
cloudCfg:	defines a configuration file for the detailed control of the kNN classification. Its details are not shown.

The maskCodeFile may look like this:

```
outsideCode       = 0
waterCode         = 20
cloudCode         = 30
noConfidenceCode  = 0
dataPresentCode   = 100
```

Output data

cloud product	Cloud mask with 1KM resolution; classCodes as specified above
basic-conf	basic confidence in 1KM resolution

Interactions

Called from *makeBasicProducts* function

3.3.3.3 Function makeSCA

This makeSCA function retrieves the snow covered area (SCA) from the image data using the NLR algorithm.

The SCA product is retrieved from band 1 in a calibrated MODIS image of any resolution.

Input data

The input directory contains ENVI files produced by the data import module.

data file	should represent a MODIS scene of any type should contain MODIS band 1 as its first band
cloud file	a result from the function *makeCloudCover*, residing in the output directory
calib file:	residing in the static directory, defining training areas in a map geometry
index file:	residing in a sub-catalogue in the output directory, required for linking the image to identification of training areas, which are defined in the map geometry
basic-conf file,	a result from the function *makeCloudCover*, residing in the output directory, required for producing the SCA-confidence

Configuration data

scaResol: defines the resolution for the SCA product
scaCfg: defines a configuration file specific for the makeSCA function.

The scaCfg file contains links to various calibration masks and identifies the methods to be used, as shown in the following lines.

scaMethod identifies the method to use, default is 'nlr', currently no other options

nlr_threshold_file: identifies a configuration file specific for the NLR method. Details are not shown here.

calibMask_250 calibration mask to be applied by the NLR method if scaResol is 250m,
calibMask_1KM: calibration mask to be applied by the NLR method if scaResol is 1km

Output data

SCA product file: basic product with SCA values with no masking from confidence, cloud cover or any other mask

SCA-conf file: confidence for basic SCA product

Interactions

Called from *makeBasicProducts* function

3.3.3.4 Function makeSTS

This function retrieves the surface temperature (STS) of snow from calibrated thermal MODIS data. The STS product is produced by means of Key's algorithm.

Input data

The input directory contains ENVI files produced by the data import module.

data file : should represent a MODIS scene of type 1KM. should contain the MODIS bands 31 and 32 as its first two layers.

angle file: should represent the view angle. It is used in Key's algorithm

Configuration data

stsMethod: identifying what method to use, default is 'Key', no other options yet
keyFileName: identifies a file with the complete set of coefficients for Key's algorithm

The rest of the parameters aim at identifying what subset of Key's coefficients to use

Output data

STS product file basic product with STS values with no masking from confidence, cloud cover or any other mask

STS-conf file confidence for basic SCA product

Interactions

Called from *makeBasicProducts*.

3.3.3.5 Function makeSGS

This function retrieves a snow grain size index (SGS) from the image data

Input data

The input directory contains ENVI files produced by the data import module.

data file:	should represent a MODIS scene of type 1KM. should contain the MODIS bands 1 and 5 as its first two layers.
latlon file:	forwarded from input catalogue if required

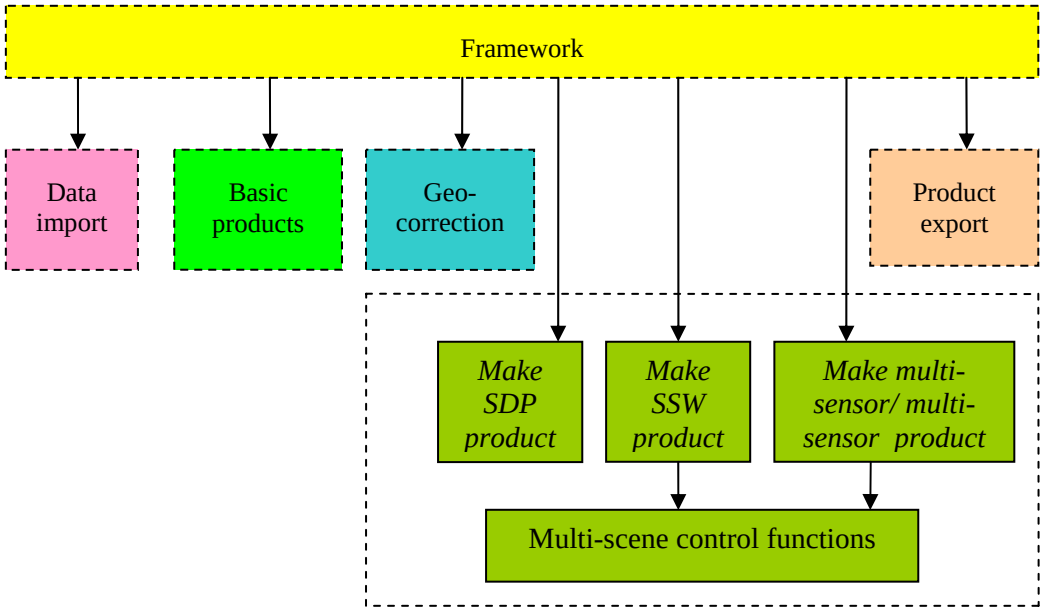
Output data

SGS product file	basic product withSGS values with no masking from confidence, cloud cover or any other mask
latlon file:	should follow the processing chain if data file is not resampled

Interactions

Called from *makeBasicProducts*

3.3.4 Module derived products



This module handles products where one needs to consider a time-series of one or more basic products. All functions in this module require that all input products are in a common geometric reference. The functions also require a cloud mask or a confidence map for each input product

All functions need to consider the time sequence of various basic products through the production chain. The framework itself only controls the various steps for one particular scene at the time, and does not know anything about the other scenes in a time-series. Therefore the multi-functions need a common toolbox, here referred to as a multi-scene controller or a time-series controller. This controller keeps track of the current time-series, and gives access to scenes that belong to the current time series.

3.3.4.1 Function makeSDP

This function estimates an index for snow surface wetness (SSW) based on the current change in SGS in addition to the current value of STS

Input data

The input consists of geo-corrected ENVI product files

- SCA product file for the current day
- Cloud cover product file

Empirical snow distribution model for the local area

Output data

SDP product for the current day

Interactions

Called from framework

3.3.4.2 Function makeSSW

This function estimates an index for snow surface wetness (SSW) based on the current change in SGS in addition to the current value of STS

Input data

The input consists of geo-corrected ENVI product files

- STS product file
- SGS product files in a short time-series
- SCA product file
- Cloud cover product file

Output data

SSW product for the current day

Interactions

Called from framework

3.3.4.3 Function makeMultiSceneSCA

This module will consider current SCA results within a running time frame and identify the best observation within that period. The required inputs are SCA products and their corresponding confidence products.

Input data

The input consists of geo-corrected ENVI product files

- sca product files in a time series
- sca-confidence files that corresponds to each sca file

Output data

SCA product file for the current day

Interactions

Called from framework

3.3.4.4 Function makeMultiSensorSCA

Input data

The input consists of geo-corrected ENVI product files

- sca product files from various sensors in a time series
- sca-conf files that corresponds to each sca file

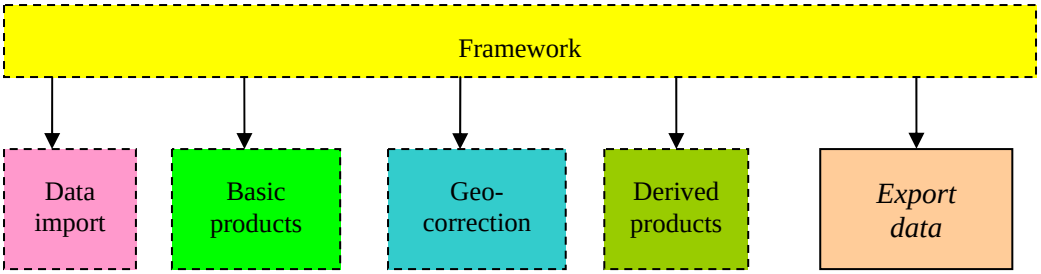
Output data

sca product file for the current day

Interactions

Called from framework

3.3.5 Module data export



The module data export contains the function *exportData*, which combines basic products or derived products with various masks in order to make a presentable final product. It also converts ENVI files to other file formats, if requested. It contains one function that can be called from the framework:

3.3.5.1 Function *exportData*

The current version of *exportData* considers one product and exports it according to specifications in the configuration data.

Input data

The input catalogue must contain a file with the specified basic or derived product.

Configuration data

The configuration files should specify these parameters:

product :	what product to export; default is 'sca'
dataOffset:	offset value to add to value of geophysical parameter; default = 0
landMask_250 :	landmask to be included if imgType is 250
landMask_1KM :	landmask to be included if imgType is 1KM
coltab	text file with colour table to be applied in the exported product

In addition the class (mask) codes in the input and output files should be specified.

Output data

The raw products are combined with the corresponding cloud mask and the static land/water mask into a presentable result. The result may be gives as ENVI-files and/or tiff-files in gray tones and/or colours, as well as a jpg-file or png-file in colours.

4 Appendix

4.1 Configuration files

The configuration file should contain information needed by the controller to perform the right operations on the different datasets. The general format of the configuration file is as follows. It should consist of keyword-value pairs: <keyword> = <value>. (The keywords currently defined are marked with bold font in the example below). The keyword should consist of one or more strings and be separated from the value with the '=' sign. The value may consist of either a single value which can be one or more strings ended by EOL, or it can be a list. The start and end of the list should be marked by parentheses, and each item should be separated by a comma: {<item1>, <item2>, <item3>}. Each item may consist of one or more strings, and the list may run over several lines. Comments should start with an '#' and end at EOL.

4.1.1 The main configuration file

In the following example, the format of the configuration file is described in more detail:

```
# ------------------------------------------------------------------------
#                  CONFIGURATION FILE
#          Production line for cryospheric variables
# ------------------------------------------------------------------------
project = cryo               # Name of project. Default value: 'cryo'
maxLog  = 100000             # Maximum size of logfile in bytes
                             # Default value: 100KB

image catalog = /nr/project/bild/images
                              # Must be an existing directory!
                              # Default value: $CRYO_TOP/data

# The following parameters must be set - there are no default values

# Specify the number of (external) providers and their ID.
nof providers = 2             # Number of data providers
providers = {ksat, nasa}      # Names of providers.
                              # For download of data over ftp, there
                              # will also need to be one ftp-configuration
                              # file per provider. The format of this will
                              # be explained below.

# For each provider, specify the mail address of the sender
# (There should only be one sender per provider)
ksat sender = mailer@ksat.no
nasa sender = mailer@nasa.no

#Specify the number of datatypes and their ID.
nof datatypes = 2                       # Number of datatypes
datatypes = { modisKsat, modisNasa}     # Names of datatypes

# Specify the number of steps in the processing chain for the datatypes.
# Syntax:    <datatype name> nof steps = <nof steps>
# There should be one processing chain for each data type.

modisKsat nof steps = 4
```

```
# Specification of steps in the processing chain.
# Syntax:  <datatype name> steps = { … }
# There should be one line for each step, where each step
# will correspond to a function in the “method API”
# Each line should be comma separated. The syntax of each line is:
#  <name of operation> <name of function> <dataset in> <dataset out> <cfg >
# Where:
# <name of operation> - the name which will appear in the GUI
# <name of function>  - the corresponding name of the function to be called
# <dataset in>        - a number specifying the dataset which is input
# <dataset out>       - a number specifying the dataset which is output
# <cfg>               - name of a configuration file for the function.
#                       If none is needed a dummy name should be given.
#                       (The file could be located under the static-
#                        directory)

modisKsat steps = {
import            import_modisdata        1    2   ksat_import.cfg,
basic             basic_products          2    3   basic_products.cfg,
project           project_modisproduct    3    4   project_products.cfg,
export            export_modisproduct     4    5   sca_export_ksat.cfg
}
# Here the datasets which are input and output should be given a number as
# an ID, with the numbering starting from 1. This corresponds to dataset
# number 1, which will always be the original raw data fetched either
# remotely or locally. Hence, in the example above, the original dataset
# will be input to convert, and the convert function will put the output in
# the directory for dataset number 2. Dataset number 2 will then be input
# to preprocess, which puts the results in dataset 3. The extraction reads
# from dataset 3 and puts the result in dataset 4. Finally, a display
# function is used to show the results in dataset 4. This function gives no
# output, and the output dataset is therefore also set to 4.

# Specification of steps in the modis processing chain.
modisNasa nof steps = 4
modisNasa steps = {
import            import_modisdata        1    2   nasa_import.cfg,
basic             basic_products          2    3   basic_products.cfg,
project           project_modisproduct    3    4   project_products.cfg,
export            export_modisproduct     4    5   sca_export_ksat.cfg
}
# Specification of local mail. (Needed for automatic remote download.)
mailusr  = cryo              # local mail user
mailpwd  = cryo              # local mail password
mailhost = mail.nr.no        # local mail host
mailport = 100               # local mail port
```

4.1.2 Configuration files for each step in the production chain

4.1.2.1 Import configuration

The configuration file *ksat_import.cfg* will identify the various image types (resolutions) to import and refer to an image type specific configuration file.

ksat_import.cfg:

```
imgTypes = { 1KM, 250  }
250_import = modis_Q_import.cfg
1KM_import = modis_cloud_import.cfg
```

Each image type specific configuration file will contain info of what parts of the image to import

modis_cloud_import.cfg

```
# Information on file and bands to be read from MOD021KM
imgType = 1KM
bandnames =     { 1, 4, 6, 19, 20, 26, 31, 32}
#
#Also read the view-angle
angleFields = {SensorZenith}
#
#Also read the lat-lon fields
lat_lon = 1
```

modis_Q_import.cfg:

```
# Information on file and bands to be read from MOD02QKM
imgType = 250
bandnames =     { 1 , 2}
#
#Also read the lat-lon fields
lat_lon = 1
```

4.1.2.2 Basic products configuration

The configuration file *basic_products.cfg* looks like this:

```
# Common parameters
maskCodeFile = mask_codes.cfg
#
# Product specific config files
cloudCfg = modis_cloud.cfg
scaCfg   = sca_compute_nlr_modis.cfg
stsCfg   = sts.cfg
sgsCfg   = sgs.cfg
#
map_1KM.cfg = data1KM_projection.cfg
map_250.cfg = data250_projection.cfg
```

The cloud detection config file defines the kNN method and the code mapping to be applied after the kNN program :

modis_cloud.cfg:

```
ClassProgramName = KNNclass
CodebookFileName = codebooks/cloudclass
KNN_val_kn = 4
KNN_val_minratio = 0.0
#
knnCodeFile      = codebooks/knn_codes.cfg
```

The map projection config file should refer to the map projection of the calibration area mask.

data250_projection.cfg:

```
inputProduct = data                    # default
inputType    = 250
projCfg      = proj_scandinavia.cfg
```

data1KM_projection.cfg:

```
inputProduct = data                    # default
inputType    = 1KM
projCfg      = proj_scandinavia.cfg
```

proj_scandinavia.cfg

```
# Projection information
# Available projections are:  UTM, POLAR
```

```
Projection = UTM
UtmZone   = 33
Datum = WGS-84
#
# Avoid '_' in ProjectionName
ProjectionName = scandinavia
#
# Polarview
UpperLeft   = { -76000.0,  7942000.0 }
LowerRight = { 1124000.0,  6392000.0 }
```

The SCA config file will identify the method and required calibration areas mask

sca_compute_nlr_modis.cfg

```
# Configuration file for SCA from MODIS images
#-----------------------------------------------
#
scaResol = 1KM
# scaResol = 250
#
# Calibration masks in UTM 33 for each resolution
# calibMask      = calib/norge_calib_33
calibMask_250  = calib/norge_calib_250m_33
calibMask_1KM  = calib/norge_calib_33
#
nlr_threshold_file = nlr_thresholds.cfg
```

The SCA-NLR config file *nlr_thresholds.cfg* should not be changed as it refers to threshold parameters that have been carefully calibrated for the current satellite sensor.

The STS config file defines the Key algorithm:

```
stsMethod   = Key
keyFileName = sts/KeyCoeffecients.txt
#AVHRR       = 16
AVHRR        = 0
MODIS        = 1
tempRange    = 2
arctic       = 1
```

4.1.2.3 Project configuration

The project configuration file will identify how to project each product

project_products.cfg

```
products = { sca, cloud}
projCfg  = proj_scandinavia.cfg
cell size   = 250

sca_resampMet   = bl
cloud_resampMet = nn
sts_resampMet   = bl
sgs_resampMet   = bl

sca_conf = 1
```

4.1.2.4 Export configuration

The export configuration file will identify how to project each product

sca_export_ksat.cfg:

```
# Configuration file for exporting SCA-product from MODIS
#-------------------------------------------------------------
product          = sca
#
dataPresentCode  = 100
noProduct        = 111
#
# dataOffset     = 0    default
dataOffset       = 100
#
landMask_250 = Envisnow_mask250_norge_z33
#
# Cloud value for NR cloud mask
in_cloud = 30
in_outside = 0
#
# Water value for NoRut vegetation mask or land/water mask
in_water = 20
#
# Code values for NR/NORUT sca product
out_cloud = 30
out_outside = 10
out_water = 20
#
coltab    = col/NVE_sca_col.txt
#
# metaFile = hdr  ==> .meta-file to be included in .hdr-file
metaFile = hdr
```

5 References

Andersen, T. 1982, "Operational snow mapping by satellites," Hydrological aspects of alpine and high mountain areas, Proceedings of the Exeter symposium, July 1982, IAHS publ. no. 138, pp. 149-154.

Amlien, J and Solberg, R, 2004. “Evaluation of algorithms for the retrieval of snow surface temperature from medium resolution satellite data”. The 8th Circumpolar Symposium on Remote Sensing of Polar Environments, Chamonix, France, 08-12 June, 2004.

Key, J.R., J. B. Collins, C. Fowler, and R. S. Stone, 1997. “High-latitude surface temperature estimates from thermal satellite data”, *Remote Sensing of Environment*, 1997. 61(2), pp. 302-309.

Koren, H, Solberg, R and Amlien, J. 2004. “Evaluation of algorithms for the retrieval of snow grain size from optical satellite data”. The 8th Circumpolar Symposium on Remote Sensing of Polar Environments, Chamonix, France, 08-12 June, 2004.

Malnes, E, Storvold, R, Lauknes, I; Solberg, R; Amlien, J and Koren, H, 2005 “Multi-sensor monitoring of snow parameters in Nordic mountainous areas” IEEE International Geoscience and Remote Sensing Symposium (IGARSS 2005), Seoul, Korea, 25-29 July 2

Solberg, R. and T. Andersen, 1994. “An automatic system for operational snow-cover monitoring in the Norwegian mountain regions,” Geoscience and Remote Sensing Symposium (IGARSS), Pasadena, California, USA, 1994.

Solberg, R, Amlien, J, Koren, H, Eikvil, L, Malnes, E, and Storvoll, R. 2004a. Multi-sensor and time-series approaches for monitoring of snow parameters. IEEE International Geoscience and Remote Sensing 2004

Solberg, R, Amlien, J, Koren, H, Eikvil, L, Malnes, E and Storvold, R, 2004b. “Multi -sensor/multi-temporal analysis of ENVISAT data for snow monitoring” ESA ENVISAT & ERS Symposium, Salzburg, Austria, September 06-10, 2004.

Solberg, R, J Amlien, H Koren, E Malnes and R Storvold 2005, “Multi-sensor multi-temporal snow cover area algorithms. Part 1: Mountain regions ” Envisnow EVG1-CT-2001-00052. Norut, Feb. 2005.

Solberg, R, Amlien, J, Koren, H, Eikvil, L, Malnes, E and Storvold, R 2005b “Multi-sensor/multi-temporal approaches for snow cover area monitoring” EARSeL LIS-SIG Workshop, Berne, February 21-23, 2005.