A generic differential interferometric SAR processing system, with applications to land subsidence and snow-water equivalent retrieval

Yngvar Larsen, Geir Engen, Tom Rune Lauknes, Eirik Malnes, Kjell Arild Høgda

Norut IT, P.O. Box 6434, NO-9294 Tromsø, Norway

ABSTRACT

Implementation of a repeat-pass differential interferometric SAR (D-InSAR) processing facility is a challenging task, involving among other things a precise coregistration of two (or more) complex SAR images, removal of the contribution from topography using a Digital Elevation Model (DEM), and band-pass filtering of common bandwidth to reduce out-of-band noise. Other useful functionality includes multi-looking, identification of layover and shadow areas, and geocoding.

In this paper we describe the design and implementation of a flexible and fully automatic D-InSAR processing facility. The design is generic, in the sense that the only part that depends on the actual SAR mission is the data reader. The data readers are designed to implement a generic interface to data, as well as necessary metadata retrieved from data headers or auxiliary metadata files.

We will also present examples from different applications based on the processing system, including use of the ubiquitous ASAR data set showing land subsidence due to the earthquake disaster in Bam, long-term subsidence in Oslo based on ERS data, as well as results from an experiment performed to retrieve the snow-water equivalent (SWE) in a Norwegian hydropower catchment from ASAR data.

1 INTRODUCTION

In a scientific setting, where data from a multitude of SAR instruments must be handled, the need for flexible processing software is imminent. Since most of the actual data processing is the same regardless of instrument, there is great potential for design of generic processing lines.

The most obvious nongeneric part of a processing line is the data reader. Every SAR instrument has its own data format, and its own way of representing metainformation. The first step towards a generic processing line is thus to identify a minimum set of metadata necessary for SAR processing. Next, a way to encapsulate access to data and metadata in a generic interface must be designed. To achieve this in a practical way, we may choose a programming language that supports object oriented programming. Preferably one where function calls may be resolved at run-time, a mechanism often referred to as *dynamic binding*. In this way we may easily hide the data source in a generic class where the actual data access is implemented by sensor specific subclasses, a concept known as *polymorphism*.

Traditional statically compiled programming languages usually resolve function calls at compile time, known as *static binding*. Some statically compiled languages also support dynamic binding, e.g. virtual functions in C++. However, another important aspect of scientific programming is flexible, rapid prototyping. This aspect is often much more important than run-time efficiency. For this reason, the remote sensing community tend to prefer interactive programming languages like Matlab or IDL which both provide high-level functions for signal processing tasks. In this paper, we present the design of GSAR, a generic processing system for SAR processing, implemented in the IDL programming language. Furthermore, we present the implementation of a differential interferometric SAR (D-InSAR) processing line using GSAR, as well as a few example results.

2 A GENERIC SAR PROCESSING SYSTEM

2.1 Base system

The system is implemented in object oriented IDL. IDL objects are persistent heap variables, and the IDL object system supports polymorphism and multiple inheritance. Internal state is encapsulated, such that access to state is only provided

if get and set methods are explicitly provided.

The base system of GSAR consists of three kinds of objects responsible for the data flow:

(1) **Transformers** are the basic units of GSAR. A GSAR transformer object is instantiated with a reference to another transformer object as input argument. In this way, we may implement a processing line by connecting GSAR transformer objects that each are responsible for one step in the data processing.
(2) **Readers** are a special kind of transformer responsible for implementing a generic interface to the data source. Generally, they are instantiated with a data location reference, e.g. a filename, as input argument.
(3) **Containers** are a special kind of transformer that are instantiated with references to *several* other transformer object as input arguments. This is needed when we want to create a single multichannel data object with data from different sources, usually different files on disk.

Note that the three different kinds of objects differ in the way they access data, but not necessarily in the way they handle the data. Thus, we may implement most of the functionality of all three transformer types in a single base class.

2.1.1 Data flow

SAR processing involves huge amounts of data. However, we are usually not able to fit all the data in computer memory at the same time. Therefore, we have to process smaller data blocks at a time.

We have implemented the data flow based on subblocks of the complete data matrix. A subblock is represented by 4 integers `xSize`, `ySize`, `xPos` and `yPos`, representing the size of the block in x and y direction and the x and y position of the first pixel of the block in the complete data matrix, respectively. The data flow is implemented through a polymorphic method `getData` (with the 4 integers representing a subblock as arguments) that operates by simply calling the `getData` method of the input object reference, and performing further data processing on the subblock returned, effectively implementing a chain of data filters.

A transformer may contain data from more than one source, for instance data from several SAR scenes from the same imaging geometry acquired at different times. We call such parallel data sources *channels*, and index them by using a keyword argument to `getData` and other per-channel methods.

2.1.2 Metadata

When a transformer object is instantiated, an internal metadata structure is formed, either by reading the necessary information from disk (reader objects), or by modifying the metadata structure acquired from the input object reference. To achieve maximum flexibility, this metadata structure is represented internally as a pointer to a pointer array of IDL structures; one structure for each channel. Both setter (`setInfo`) and getter (`getInfo`) methods are implemented.

2.1.3 File output

Any useful data processing software must include a way to write results to disk in some format. We have chosen to use a common binary format for the data, namely band sequential (BSQ). A reader for this format is easily written in any programming language. The metadata structures are written to an XML file. This metadata includes at least the following information: dimensions of each channel, endian, IDL datatype (float, double, complex, integer, byte, etc.), and number of channels. We call the image format Generic IMage Format (GIMF).

A typical transformer applies a filtering function to the data block. Thus, to avoid edge effects, there is often a need to remove a number of pixels at the edges of the block. File output is implemented by a method `write` that processes each channel of the transformer sequentially by partitioning the channel in subblocks with the correct overlap to account for removal of the edges.

Implementation of a different output format, e.g. geoTIFF, is easily achieved by simply writing a new transformer that inherits the base class and modifies the polymorphic `write` method to implement the new output format. Thus, such a transformer may be used as the last element of any processing line.

2.1.4 State and settings

A transformer object may have internal state settable from outside. For instance, the `write` method needs to know the size of the processing blocks as well as the number of pixels to remove at the edges of each processed block in order to calculate a partitioning of the data matrix with correct overlap between blocks. Manipulation of internal state is implemented through the polymorphic methods `get` and `set`.

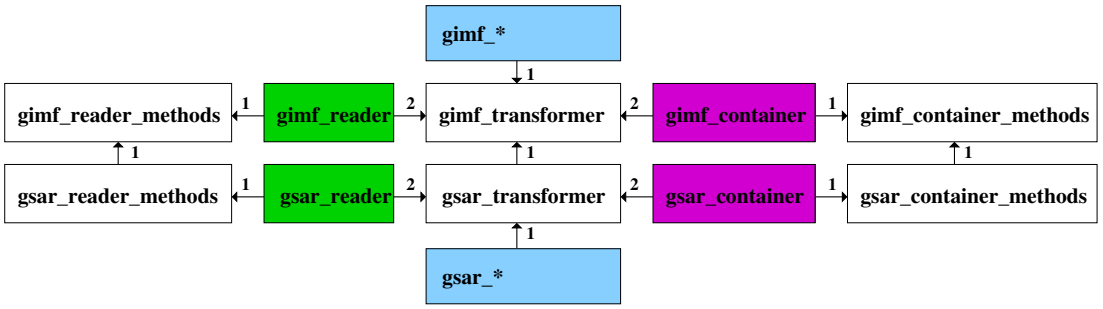

Figure 1. Inheritance graph for the base system of GSAR. The numbers denote inheritance order.

2.1.5 SAR specific functionality

All the functionality outlined so far in this section are not SAR specific, but applies to any image processing. Thus, we have chosen to split the base class in two: one class `gimf_transformer` implementing the infrastructure mentioned so far in this section, and one class `gsar_transformer` which inherits `gimf_transformer`, and in addition implements the SAR specific parts, including the interface to an orbit propagator, as well as geocoding functionality [4]. This functionality includes

- A method `getStateVector` that interfaces a properly initialized orbit propagator (or possibly interpolates precalculated statevectors if available).
- A method `getGeometry` that returns an IDL structure containing the following information about a given position in the data block (or a given latitude/longitude/height within the data block): two-way time, absolute azimuth time, latitude and longitude, incidence angle, look angle, ground track velocity, satellite velocity, track angle relative to north, target 3D position in earth centered earth fixed (ECEF) coordinates, earth norm in ECEF, satellite statevector at zero-Doppler time corresponding to position, and relative position, velocity and acceleration between satellite and target in earth centered inertial (ECI) coordinates. The WGS-84 ellipsoidal earth model is used.
- A method `getDoppler` that returns the Doppler centroid at a given position in the data block. The base class method just retrieves this by interpolating in a grid stored in an internal variable. The reader object has the responsibility to initialize this grid, either by reading precalculated estimates already contained in the data, or by estimating from data.

Almost any kind of SAR processing line may be implemented based on the functionality implemented by the base class `gsar_transformer`. In fact all the three kinds of transformers mentioned in Section 2.1 may be implemented by inheriting the `gsar_transformer` and implementing the `init/cleanUp` functions to do any necessary preprocessing or acquisition of auxiliary data, and then implementing the `getData` function. This is done slightly differently for the three kinds of transformers.

- A **transformer** implements `getData` by retrieving a data block by calling the `getData` method of the object supplied as input argument, and applying further processing on it. A different transformer is needed for each kind of data processing step, but the transformer classes usually do need any instrument specific information.
- A **reader** implements `getData` by simply reading a data block from a file. A different reader is needed for each kind of external data. Thus, the readers are the only nongeneric classes.
- The **container** implements `getData` by dispatching on the channel argument and calling the `getData` method of the correct of several input objects. This object behaves exactly as an ordinary transformer object. The only difference is that the input data channels are retrieved from several generic objects instead of one. Thus only one such class is needed, and it is implemented as `gsar_container`.

The complete inheritance graph of the GSAR base system (including the GIMF subsystem) in shown in Fig. 1. The generic reader (a reader of the GSAR external format) is shown in green, the container is shown in magenta, and the transformers are shown in blue. note that all three inherit from the `gsar_transformer` base class.

2.1.6 Summary of generic transformer methods

Due to multiple inheritance property, all the methods described above is automatically available in *all* GSAR transformer objects.

- Data flow is implemented in a transformer by modifying the following polymorphic base class method
  ```
  function gimf_transformer::getData, xSize, ySize, xPos, yPos, $
     channel=ch, Fourier=f
  ```
  The arguments are four integers representing a subblock, a keyword argument representing the channel number (default: 0), and a keyword flag telling whether the data should be returned in the Fourier space or not. The method returns a

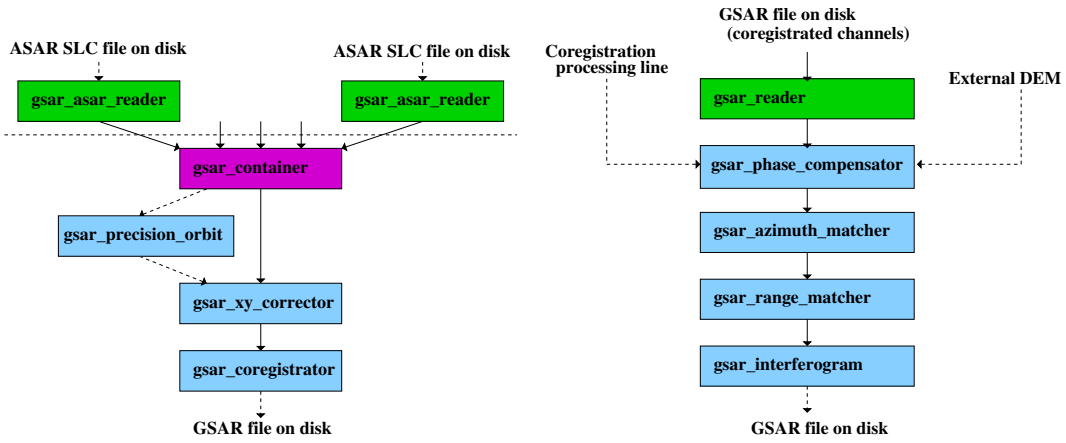

Figure 2. Differential InSAR processing chain. Left: Coregistration of ASAR SLC images. Right: interferometric processing.

datablock of size `xSize` by `ySize`.

- Metadata access is implemented in a transformer by modifying the following polymorphic base class methods
  `function gimf_transformer::getInfo, channel=ch`
  `pro gimf_transformer::setInfo, info, channel=ch`
  These methods gets and sets, respectively, an IDL structure containing metadata.
- Access to state and settings is implemented in a transformer by modifying the following polymorphic base class methods
  `function gimf_transformer::get, channel=channel`
  `pro gimf_transformer::set, channel=channel, _extra=extra`
  These methods gets and sets, respectively, an IDL structure containing state and settings. The `_extra` keyword is very useful in a method in a base class that is meant to be polymorphic, since this captures any keywords not explicitly defined dynamically. Thus, each transformer may implement setting and getting of its own state and pass the rest on to the preceding transformer in the processing chain through the `_extra` keyword.
- File output is implemented in a transformer by modifying the following polymorphic base class method
  `pro gimf_transformer::write, filename, xSize, ySize, xPos, yPos, channels=chs`
  This method writes a processed subblock (default: all data) from the given channels (default: all channels) to file in the generic GIMF format (BSQ + XML).
- The following base class methods are inherited by any GSAR transformer.
  `function gsar_transformer::getStateVector, yTime, channel=ch`
  `function gsar_transformer::getGeometry, xPos, yPos, $`
  `  geographic=geo, height=h, channel=ch`
  These methods return geographical information about a given pixel position in the data, or a given latitude/longitude/height within the data (height defaults to 0 if no DEM is available).
  `function gsar_transformer::getDoppler, xPos, yPos, channel=ch`
  Access to Doppler centroid estimates is implemented by the following polymorphic base class method. It may be implemented either by reading estimates from external metadata, or by estimating the Doppler centroid from data, depending on the data source.

3 DIFFERENTIAL INTERFEROMETRIC SAR PROCESSING

Differential interferometric processing consists of 2 steps. The first is to precisely coregister a number of images of the same area acquired at different times. The second is the interferometric processing, including the differential part: removal of interferometric phase due to topography and earth curvature.

The GSAR coregistration chain is shown in the left part of Fig.2. The `gsar_xy_corrector` object uses `getGeometry` to coarsely identify the area common to all channels, and then finds the average pixel shift between the master channel (usually channel 0, but configurable) and each of the other channels (slaves) by cross-correlation, see e.g. [2,4]. The calculated shift is very accurate since it is averaged over a lot of estimation windows distributed across the entire scene. No ground control points are needed. Since we use an orbit propagator that is initialized using a single statevector to calculate satellite statevectors across the scene, we can account for most of the orbit error by simply moving the initialization statevector of the slaves by an amount corresponding to the shifts observed.

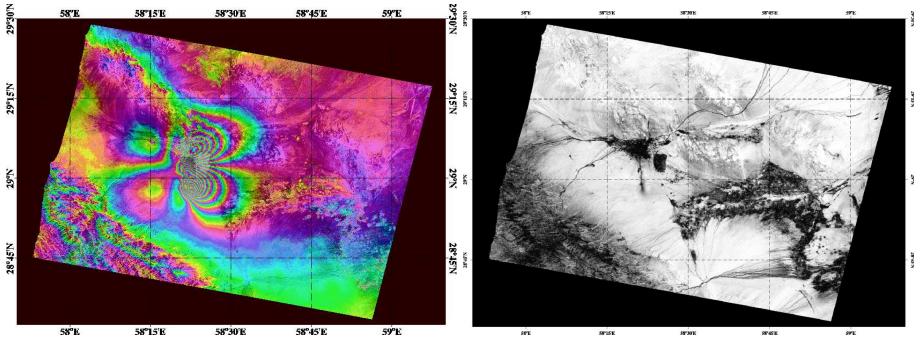

Figure 3. Top: 2004-02-11 vs. 2003-12-03, 2m baseline. Coseismic. Left: Differential interferometric phase. Right: coherence. Data courtesy of ESA.

The `gsar_coregistrator` object accounts for possible skew in subpixel shifts across the scenes caused by nonparallel orbits, or satellites that are not steered precisely, such that the squint is varying a lot for scenes acquired in the same geometry. This is done by modeling the variation in subpixel shift across the scenes as linear in azimuth and range, estimating the rates of change, and correcting for these using a chirp-z transform [7]. The factor in azimuth direction is expected to be very small, and may be skipped by setting a flag with `set`, but the effect in range may be significant in some cases.

The optional `gsar_precision_orbit` substitutes the satellite statevector given in the data file with a postprocessed precise orbit statevector of some kind, in this case the ENVISAT precise orbits (EIGEN-GRACE01S) provided by Delft Institute for Earth-oriented Space Research. Notice that though chain in the figure is for ASAR SLC files, everything below the dashed line is generic (except the precision orbit object), such that to do the same for other sensors, only the readers above the dashed line must be changed.

The GSAR interferogram formation chain is shown in the right part of Fig.2. The input is either from a GSAR file containing coregistered channels, or directly from the coregistration chain described above (slower).

The `gsar_phase_compensator` removes the flat earth phase and the interferometric phase component due to topography (if an external DEM is available) as outlined in [1]. The `gsar_azimuth_matcher` and the `gsar_range_matcher` filters out the nonoverlapping bandwidth in azimuth and range [3], respectively. Finally, the interferogram itself is formed by `gsar_interferogram`, which upsamples the two SLC channels, multiplies, and finally applies a lowpass filter before downsampling in order to avoid aliasing. Multilooking in range and azimuth are settable options of this transformer.

4 SELECTED RESULTS

In this Section, we will briefly show the results from some applications using GSAR processed interferograms.

4.1 Land subsidence

4.1.1 Bam earthquake, December 26, 2003

The final differential interferogram product of the processing chain described in Section 3 contains information on land subsidence. The well known coseismic data set from the Bam earthquake made available by ESA is a good litmus test for any interferogram software due to its extraordinary high temporal coherence, combined with a thoroughly analyzed subsidence pattern due to the earthquake. Fig. 3 shows results from applying the GSAR interferometric processing line to the Bam data set. The 3 arcsec SRTM DEM is used for removal of interferometric phase, and we have used multilooking factors of 2 in range and 8 in azimuth. A small amount of postprocessing, including geocoding and removal of a linear phase ramp in range and azimuth due to orbit errors. There are still some atmospheric effects left in the result.

4.1.2 Urban subsidence in Oslo, Norway, 1992-2000.

Fig. 4 shows results from applying the Small Baseline Subset technique to a stack of GSAR processed ERS interferograms from Oslo, Norway, 1992-2000. For details, see [6]. Note in particular the highly nonlinear behavior of the deformation time series in the rightmost panel. This is due depletion of ground water caused by the construction of a tunnel for the airport train right beneath this point.

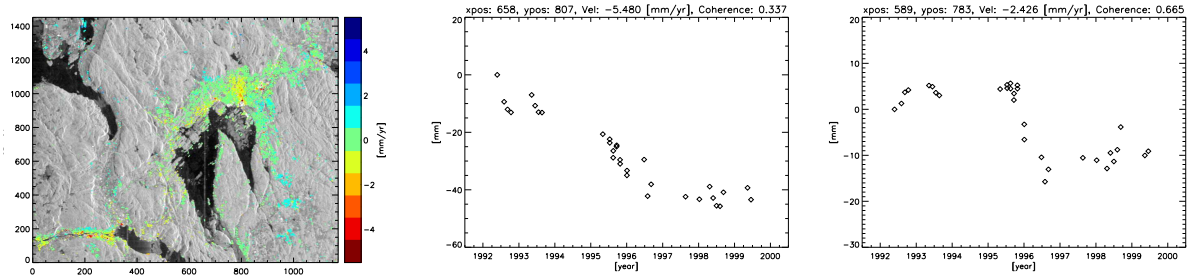

Figure 4. Left: Mean subsidence rate in Oslo area 1992-2000. Right: Subsidence time series for two points in the Oslo area. (Bjørvika and Hellerud.)

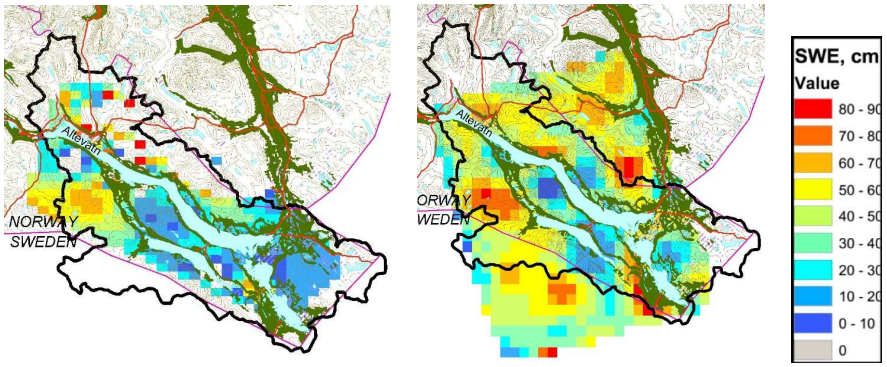

Figure 5. SWE results for Altevann, Norway, Mar/Apr 2004. Left: Interpolated in-situ measurements (courtesy of Statkraft). Right: SWE estimated from ENVISAT ASAR by the SWEDeK method.

4.2 SWE retrieval by delta-K D-InSAR

As a final example, we show the results from using GSAR processed interferograms for retrieval of Snow Water Equivalent (SWE) from ENVISAT ASAR data near a catchment for a hydropower dam in Northern Norway. For details, see [5]. SWE retrieval is of key interest in the hydropower industry. Currently, expensive and time consuming manual measurement campaigns are used, and the coverage is naturally not good. Thus, SWE retrieval by SAR is of high commercial interest. Fig. 5 shows interpolated in-situ measurements (left), and corresponding estimated results from ENVISAT ASAR.

References

[1] M. Eineder. Efficient simulation of SAR interferograms of large areas and of rugged terrain. *IEEE Transactions on Geoscience and Remote Sensing*, 41(6):1415–1427, 2003.

[2] A. K. Gabriel and R. M. Goldstein. Crossed orbit interferometry: theory and experimental results from SIR-B. *Int. .J. Remote Sensing*, 9(5):857–872, 1988.

[3] F. Gatelli, A. Monti Guarnieri, F. Parizzi, P. Pasquali, C. Prati, and F. Rocca. The wavenumber shift in SAR interferometry. *IEEE Transactions on Geoscience and Remote Sensing*, 32:855–865, July 1994.

[4] H. Johnsen, I. Lauknes, and T. Guneriussen. Geocoding of fast-delivery SAR image mode product using DEM data. *Int. J. Remote Sensing*, 16:1957–1968, 1996.

[5] Y. Larsen, E. Malnes, and G. Engen. Retrieval of snow water equivalent with Envisat ASAR in a Norwegian hydropower catchment. In *Proc. IGARSS-2005, 25–29 July*, 2005.

[6] T. R. Lauknes, J. Dehls, K. A. Høgda Y. Larsen, and D. J. Weydahl. A comparison of SBAS and PS ERS InSAR for subsidence monitoring in Oslo, Norway. In *Proc. FRINGE-2005 Workshop, Frascati, 28 Nov–2 Dec*, 2005.

[7] L. R. Rabiner, R. W. Schafer, and C. M. Rader. The chirp z-transform algorithm. *IEEE Trans. Audio Electroacoustics*, 17(2):86–92, 1969.