SANDIA REPORT
SAND2013-6567
Unlimited Release
Printed August 2013

MREG V1.1: A Multi-Scale Image Registration Algorithm for SAR Applications

Paul H. Eichel

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

SAND2013-6567
Unlimited Release
Printed August 2013

MREG V1.1: A Multi-Scale Image Registration Algorithm for SAR Applications

Paul H. Eichel
Signal Processing & Technology Department
Sandia National Laboratories
P.O. Box 5800
Albuquerque, New Mexico 87185-1207

Abstract

MREG V1.1 is the sixth generation SAR image registration algorithm developed by the Signal Processing & Technology Department for Synthetic Aperture Radar applications. Like its predecessor algorithm REGI, it employs a powerful iterative multi-scale paradigm to achieve the competing goals of sub-pixel registration accuracy and the ability to handle large initial offsets. Since it is not model based, it allows for high fidelity tracking of spatially varying terrain-induced misregistration. Since it does not rely on image domain phase, it is equally adept at coherent and non-coherent image registration. This document provides a brief history of the registration processors developed by Dept. 5962 leading up to MREG V1.1, a full description of the signal processing steps involved in the algorithm, and a user's manual with application specific recommendations for CCD, TwoColor MultiView, and SAR stereoscopy.

ACKNOWLEDGMENTS

Numerous individuals at Sandia National Laboratories have contributed in the development of SAR image registration expertise over the years. We make some effort to chronicle the major contributors in the History section of this report, but just to be unequivocal; MREG would certainly not exist without the efforts of Paul Thompson and Terry Calloway.

CONTENTS

1. INTRODUCTION 7

2. HISTORY 8
    2.1 SNL CCD.CSH 8
    2.2 SNL CCD Release 1.0 9
    2.3 SNL DYNREG 10
    2.4 SNL MSREG 10
    2.5 SNL REGI 11

3. SNL MREG 12
    3.1 MREG Preliminaries 12
    3.2 *mregister*() 15
        3.2.1 Prewarp 15
        3.2.2 Tie Point Computation 17
        3.2.3 Complex Image Warp 19
        3.2.4 Outputs 20

4. MREG User's Manual 21
    4.1 The MREG command line 21
    4.2 Command line arguments 21
    4.3 Discussion 22
        4.3.1 CCD 22
        4.3.2 TwoColor MultiView 23
        4.3.3 Stereoscopy 23

5. CONCLUSIONS 24

6. REFERENCES 25

TABLES

Table 1: SARTAGS Fields Utilized by mregister. 14

1. INTRODUCTION

MREG V1.1 represents the sixth generation in a progression of ever more powerful image registration codes developed by the Signal Processing & Technology Department (5962) for SAR applications. Researchers in the Department recognized the need for sub-pixel image registration accuracy in order to pursue pioneering research in SAR coherent pair processing. In the late 1980s and early 1990s, these research efforts culminated in many first-in-kind demonstrations of Coherent Change Detection (CCD), Interferometric terrain mapping, and Stereoscopic terrain mapping. The early registration and CCD algorithms were packaged into the CASE EXECUTIVE software release and employed by a large number of image analysts providing the first practical applications of SAR coherent pair processing.

MREG V1.1, like its predecessor REGI, is a non-coherent, multi-scale registration algorithm. Being non-coherent, it is equally adept at registering pairs of images that exhibit phase coherency (such as used for CCD) or, by virtue of less stringent collection geometries, pairs of images that lack such coherency (such as typically employed for TwoColor MultiView). Its multi-scale iterative structure allows MREG to accommodate large and spatially varying initial offsets and still produce sub-pixel registered final products. Unlike polynomial (model) based registration algorithms, no restrictions are placed on the randomness of spatial registration offsets allowing for excellent performance on image pairs exhibiting terrain-induced differential layover such as those employed for stereoscopic terrain mapping.

This document serves several functions. First, a brief history of the development of registration processors at SNL is summarized. Second, the signal processing steps are provided in a complete and concise manner with an aim to documenting the software. Finally, a user's manual is included, with specific recommendations of command line options for processing CCD, TwoColor MultiView, and Stereo.

2. HISTORY

In 1989, a nascent research project conducted by the Signal Processing & Technology Department (originally The Systems Research Division, 0315, and later The Signal Processing Research Group, 5912) received a significant windfall of data. These data consisted of numerous complex SAR images that had been collected by an airborne system (the ERIM N1) in support of another Sandia program. At the time, Div. 0315 was entertaining the notion that complex SAR imagery might be suitable for two-pass interferometry. (Single-pass Radar interferometry had been demonstrated by Goodyear Aerospace as early as 1974 [(Graham 1974)]). Not just any pair of complex SAR images would be candidates for interference phenomena, however. Theoretical work on the part of ERIM, under contract to Sandia, had suggested that tight repeat collection geometries were necessary to form mutually coherent image pairs. Since the N1 images were collected using a sophisticated beacon system to guide the aircraft to precise repeat geometries, these images appeared to be excellent candidates to kick start research into SAR interferometry.

The question remained as how a given pair of two-pass images was to be processed to yield an interference pattern. After studying the technique of Gabriel and Goldstein using SIR-B data [(Gabriel 1988)], Paul Eichel of Div. 0315 postulated that 2-D subpixel co-registration of the *image* domain datasets might yield continuous interference patterns without iteratively reforming the images from phase history. To test this theory, he wrote a small program for the department's MegaVision real-time image processing system wherein the phase of one image, discretized to one byte per pixel, was loaded into one image plane and the conjugate phase of the second image was loaded into another image plane. The pixel level difference (phase difference) was computed at video rates to a third plane and routed to the real-time display. The second image could be spatially offset in both directions under the control of the trackball. By slowly displacing image 2 relative to image 1 via the trackball, Eichel unequivocally demonstrated interference patterns throughout the footprint of the image pair, albeit with offsets that varied spatially.

Armed with this discovery, and aided by image warping expertise contributed by Dennis Ghiglia and Gary Mastin, Eichel wrote a comprehensive complex SAR image registration code described in the following section. This code was used to demonstrate the first full-scene interferometry and Coherent Change Detection (CCD) results from 2-pass airborne collections in 1989. Regarded as highly classified at the time, this technique was published only later after similar results were reported in the open literature. The source code for all of the registration algorithms discussed below has been preserved.

2.1 SNL CCD.CSH

The first generation image registration software, SNL CCD.CSH, was actually a collection of programs executed under the control of a C-shell script, *ccd.csh*. The registration methodology employed by *ccd.csh* has been fully documented in Chapter 5 of [(Jakowatz 1996)]. Here we will provide an overview. The basic notion is to compute a regular grid of control points or tie points that measure the local displacement between the pair of images. Thus, a sparse set of sub-image blocks, typically sized 64 X 64 pixels and spaced by 512 pixels, are extracted from the original complex images and compared. The local displacement for each control point block is determined by computing the complex correlation over a range of spatial offsets and finding the maximum of the correlation surface. This computation is facilitated by using FFTs.

Having found a 2-D set of control points, the local measurements are automatically edited for outliers, and the surviving control points are regressively fit to a low order 2-D polynomial. A second order fit is generally used. This 2-D polynomial in turn is used to resample (warp) one image to the other, typically achieving sub-pixel accuracy in the neighborhood of a tenth of an IPR main lobe for images with high complex coherence.

Since the search space of each control point computation is fundamentally limited by the size of the correlation block size, and since global image offsets can sometimes be much larger than this, the just described high resolution registration stage was preceded by a low-resolution, non-coherent (detected) registration stage to coarsely back out such gross offsets. The reader is referred to [(Jakowatz 1996)] for a complete discussion of this technique.

CCD.CSH was employed by Div. 0315 in many first-ever two-pass SAR interferometry and CCD experiments in 1990 and 1991. The most important of these seminal results were catalogued in [(Eichel 1993)].

2.2 SNL CCD Release 1.0

By 1991, it had become evident to researchers in Div. 0315 that polynomial based registration techniques were not suitable for every 2-pass application. In particular, where the SAR collections are obtained on significantly different ground tracks (greater than a few degrees) and in the presence of non-trivial terrain relief in the imaged scene, substantial amounts of height induced differential layover exists in the image pairs (see [(Jakowatz 1996)]). Because the differential layover is spatially varying and local in nature, low-order 2-D polynomials are ill suited to modeling the image-to-image disparity. Charles Jakowatz made the important observation that a denser set of control points followed by the computation of a non-parametric 2-D displacement surface ought to handle such situations. Eichel, again with the help of Ghiglia and Mastin, then implemented a tessellation-based image warper. This warper, in turn, was based on a triangular tessellation fitting routine written by C. L. Lawson of the Jet Propulsion Laboratory.

The sequence of operations for this method is similar to that described above and is also documented in reference [(Jakowatz 1996)]. The main differences are that a much denser set of control points are computed and these are then fit not to a 2-D polynomial, but rather comprise the vertices of a triangular tessellation. Two spline surfaces are computed: an X-displacement surface and a Y-displacement surface. The surfaces have triangular flat plates with vertices at the control point locations. Each triangle is a best spline fit of the displacements at its vertices subject to the constraint that the first partial derivatives are continuous at the boundaries (cubic spline fitting) [(Press 1992)].

The spline-fitting registration algorithm was integrated with the polynomial-based codes and released with version control as SNL CCD Release 1.0. In 1993, this version was made available outside of SNL in a third-party package called CASE EXECUTIVE. CASE EXECUTIVE was quickly adopted by a wide range of image analysts and became the standard CCD tool of the day. The code was restructured by Eichel to take advantage of the simple and efficient shared memory parallel processing architectures such as the Silicon Graphics 240-GTX. Gary Mastin ported the code to the Cray XMP architecture for a set of demanding users.

2.3 SNL DYNREG

DYNREG was the brainchild of Paul Thompson. This registration algorithm was based on a completely different paradigm than CCD Release 1.0. Rather than using a strategy of generating a grid of control points and fitting them to a surface, DYNREG employed dynamic programming. DYNREG made iterative passes through a pair of images, attempting to find patch-wise offsets that maximized the complex correlation between the images. As the algorithm progressed through the raster images, succeeding row and column offsets were predicted from present offsets by means of the dynamic programming logic. Thus, DYNREG made allowances for local, terrain-induced differential layover in a smoothly varying manner, but without either a polynomial or a tessellated surface model.

DYNREG became operational in late 1994. However, it did not enjoy widespread adoption. This may have come about not because of any lack of performance but rather because its user interface was widely perceived to be cumbersome. A sophisticated knowledge of the underlying principles of the dynamic programing engine was required in order to obtain the best performance from the algorithm. Although Thomas Flynn made various modifications to the algorithm to improve the correlation computations and make it easier to adjust search parameters, DYNREG never enjoyed the large impact that CCD Release 1.0 had made. It was added to the CASE EXECUTIVE package in 1996, and modified for parallel execution using the Message Passing Interface (MPI) protocol by Ireena Erteza in 1999-2000. The MPI version was successfully used in the Exercise Special Project 99 (SP 99), an AF TENCAP extended real-life scenario demonstration, delivering increased performance and intelligence from CCD in highly demanding scenes.

2.4 SNL MSREG

The fourth generation registration software package was also initiated by Paul Thompson. This highly effective algorithm has proven to be extremely robust and versatile. At its core, it is a multi-scale, iterative registration technique that uses a minimum least squares metric instead of complex correlation to control its search engine. Because it is non-parametric and non-coherent, it is equally effective in applications ranging from CCD to TwoColor MultiView to Stereoscopy. This algorithm has remained at the core of all subsequent registration developments in the Signal Processing and Technology Department up to and including the subject of this report, MREG. It has been successfully applied to a very large number of two-image and multi-image SAR registration problems for almost two decades.

MSREG is a classic multi-scale registration algorithm. Numerous passes are made through the image pair, starting with a very small, highly downsampled, low resolution version of the images, and progressing through higher and higher resolution stages until the last pass is made at full resolution. The accumulated registration offsets for all preceding stages are used to predict the offsets at the current stage, and the new differential updates computed are then added to the accumulator.

This strategy has a number of important advantages in the typical SAR application. Firstly, since the first stage is performed at a very low spatial resolution, very large initial offsets can be accommodated with a rather small search space (a few pixels at low resolution represents large distances in meters). Secondly, the method is non-parametric, as an exhaustive search is conducted at each resolution stage, albeit over a limited set of offsets. The search is not limited to any preconceived model; therefore completely random values and spatial distribution of differential layover may be accommodated. Thirdly, since the final search iteration is performed on the full resolution data, sub-pixel registration accuracy is readily achieved. Finally, the method achieves a high level of performance, the stages accomplishing a large, arbitrary search space with a relatively small number of computations.

2.5 SNL REGI

The SNL REGI code is not, in itself, an image registration code. Rather, it incorporates the MSREG registration engine embedded in a much larger, multi-purpose piece of software. Written largely by Terry Calloway, the REGI code is actually an end-use *application,* with complex image pairs as inputs, and a wide variety of final or intermediate products as outputs. REGI was conceived firstly as an intelligent processor for Coherent Change Detection (CCD). In addition to the basic registration / complex correlation CCD processing elements, it also has a large repertoire of automated phase compensations, both deterministic and data-driven, to help achieve the best possible CCD correlation and phase maps. It has the ability to make use of third party Digital Elevation Models (DEMs), where available, to aid this process. REGI makes rich use of image metadata, imaging geometry and frequency space characteristics of the input images, in order to (a) automatically choose optimal processing parameters, and (b) apply various signal compensations to improve performance. The code has provisions for tailoring the signal processing for numerous applications: CCD, TwoColor Multiview, Stereoscopy and DEM generation, Interferometry, Subsidence measurement, and generating Anaglyphs. Of these, only the CCD outputs are produced by the REGI code itself, but it performs most of the image pair processing steps on the path to the other products.

Since its introduction in 2005, REGI has become the workhorse code for practically all coherent pair and non-coherent pair processing tasks in the Signal Processing and Technology Department. Further, it has been embedded in large, multipurpose interactive SAR workstations such as SLOAN and CSISAR, as well as in specialized codes producing orderable products for national users such as DAILY WATCH. It has been applied to image pairs from dozens of different SAR systems, US and foreign, airborne, spaceborne, and bistatic.

3. SNL MREG

SNL MREG breaks no new ground from an algorithmic standpoint. It employs the same multi-scale registration philosophy of MSREG and REGI. In fact, the signal processing functions are largely carried over from the latter. However, for certain applications, REGI has grown too complex and all-encompassing. MREG was conceived as a more single purpose code, namely robust registration of SAR image pairs, without the internal complexity of REGI. In this way, it is something of a return to the philosophy of MSREG. On the other hand, it benefits from years of experience with the later code in incorporating a canonical parameter set for easy, effective application to real world problem sets. MSREG never achieved this level of application.

MREG has other characteristics that lend it to embedded applications. It is a clean-sheet rewrite of the multi-scale algorithm in ANSI-C. Whereas its predecessors are amalgamations of Fortran 77, Fortran 90, and C, modified and extended over the years as experience was gained, MREG is much more structured and tightly written. It has 4% the lines of code as does REGI, albeit with a simpler task to accomplish. However, in most SAR pair-product applications, the image registration portion represents the overwhelming lions' share of the total computational load. Thus, by excising this critical task from the much larger REGI code, MREG provides a critical building block toward a more modular approach in embedded environments.

Finally, the MREG source code, along with this report, provides the first detailed documentation of this very effective and versatile registration algorithm. Neither of its predecessors has been documented in any meaningful way, and their source code is somewhat obscure. Every effort was extended in the rewrite to make the source code as transparent as possible while trading away nothing in performance.

3.1 MREG Preliminaries

MREG is registration code for complex SAR images. For the purposes of this document, a complex image is represented by a two-dimensional array of pixels, each of which is comprised of two single-precision floating point values, a real and an imaginary component. Since ANSI-C does not have an intrinsic complex value data type, MREG uses a two-valued structure to define a complex variable. Thus, the storage requirements of a complex SAR image are 64 bits per pixel.

The object of a registration algorithm is to resample one of the input images such that it exactly overlays the other image. That is, if we were to examine individual pixels of the registered images at a given array position, they will coincide with the exact same ground position in the scene. Furthermore, since the images are complex, both the magnitude and phase of the pixel values of that scene position will be correctly represented. Some terminology is in order. Following widespread convention, the two input images to be registered will be referred to as the *Reference Image* and the *Mission Image*. The *Mission Image* is to be resampled to overlay the *Reference*. After resampling, the *Mission Image* becomes the *Registered Image*. Thus, the *Registered Image* contains the scene content of the *Mission Image* (at the moment in time the *Mission Image* was collected), but coincides with the sampling grid of the *Reference Image*. The registration code should therefore accept as inputs the *Reference* and *Mission* complex images (along with their associated metadata), and produce as an output the *Registered Image*. The *Reference Image* is assumed to be unchanged by the registration process.

Unlike its predecessors, MREG is designed to be an *embeddable* algorithm, not an interactive or command line executable. The controlling routine, called *mregister*(), is called with six arguments: pointers to complex arrays containing the *Reference and Mission* images, a pointer to a complex array that will hold the *Registered Image*, two pointers to structures holding the *Reference* and *Mission* image metadata, and a pointer to a structure holding MREG runtime parameters. Since the *Reference, Mission,* and *Registered* images must perforce all be resident in memory simultaneously, the memory requirements of MREG are overwhelmingly driven by these three arrays. Indeed, as will be seen, the memory required in almost all situations may be easily estimated as 3.25 times that of the *Reference Image*.

SAR image metadata always plays an important role in the registration process. Generally speaking, SAR images may be treated as a particular mapping of a portion of the earth's surface onto an *image plane*. The nature of the mapping, as well as the particulars of the *image plane* sampling grid, varies considerably from SAR system to SAR system and even among particular images of a given system. Examples of SAR *image planes* and sampling grids include *slant planes* and *ground planes*, and Range-Doppler, Cartesian, Range-Cross Range, and many other grids. Sometimes the *image plane* may not even be a plane at all; *geocoded* images produced on a map projection or a Digital Elevation Model (DEM) surface being examples. The image metadata fields inform the registration code about the particulars of its input images. Under certain very simple situations where the *Reference* and *Mission* images were deliberately imaged by the same SAR system, with the same parameters, sample grids, image planes, and viewing geometry, this metadata may not play an important role. However, in most real-world situations, the two input image parameters and viewing geometry may be significantly or radically different. MREG uses the metadata to perform a deterministic *prewarp*, discussed later, in order to avoid costly extra computation in the registration process.

Metadata is passed between routines via a structure denoted SARTAGS. The SARTAGS structure is defined in the include file *sartags.h.* This structure definition supports not only MREG but nearly all other SAR signal processing codes in Dept. 5962. It also functions as the metadata container for SAR images in the SRF file format, the native format used for all applications in the Department. The SARTAGS structure has a very comprehensive set of metadata fields. Various fields provide for descriptions of the data source, image array fields, radar geometry fields, SAR dispersed (frequency space) domain fields, geodetic fields, interferometric fields, and phase history fields. All are expressed in the most canonical manner possible, in double precision floating point, MKS units, and an ECEF coordinate system. The MREG software only requires a very small subset of these parameters, however. In particular, the following table lists the SARTAGS parameters that must be present when *mregister*() is called.

Field	Type	Units	Description
Mode	string	-	"C8" or "IQ4" denoting complex data format.
Source	string	-	Name of image source, i.e. platform.
Dim[2]	integer	-	(X,Y) dimensions of image array.
Geo_Flag	string	-	Y = Georeferenced Image; N.
Corners[8]	double	decimal degrees	Corner Coordinates: Lat, Long.
Post_sp[2]	double	decimal degrees	Sample spacing for Georeferenced Images.
FPN[3]	double	meters	Focus plane normal (ECEF).
APC[3]	double	meters	Aperture phase center (ECEF).
GRP[3]	double	meters	Scene reference point (ECEF).
Img_GRP[2]	integer	-	Array pixel (X,Y) corresponding to GRP.
SF[2]	double	meters	Pixel scale factors, (X,Y) at GRP.
IPR[3]	double	meters	Image plane unit vector, Range (ECEF).
IPCR[3]	double	meters	Image plane unit vector, Cross Range (ECEF).

Table 1: SARTAGS Fields Utilized by *mregister()*.

MREG is capable of intelligently handling georeferenced images in geographic (lat, long) coordinates. If the input images are georeferenced, the Geo_flag field must be set to "Y" and the Corners and Post_sp fields must be filled. Otherwise, the latter fields are ignored. Conversely, Img_GRP is ignored if Geo_flag = "Y".

While *mregister*() is the actual executive routine for MREG, the software does have a wrapper function, called *mreg*(), provided for command line applications. *mreg*() is the interface between the Dept. 5962 – specific processing environment and *mregister*(). As such, it provides for a command line parser, inputs the *Reference* and *Mission* images along with their metadata in SRF format, allocates memory for the *Registered Image*, calls *mregister*(), and outputs the *Registered Image* in SRF format. For command line applications using other complex SAR image formats, it would be a simple matter to duplicate these functions following the example of *mreg*().

When using the command line wrapper *mreg*(), the command parser allows the user to specify various runtime parameters. These will be discussed in the User Guide Section. The runtime parameters are passed between functions using a structure called PARGS defined in the include file *mreg.h*. Embedded applications also need to populate this structure before calling *mregister*().

3.2 *mregister*()

The function *mregister*() is the principal executive routine for MREG. As noted previously, it is called with six arguments: pointers to the *Reference*, *Mission*, and *Registered* complex images, pointers to the input images' metadata structures, and a pointer to the runtime parameter structure. The function returns with the *Registered Image* placed in the array pointed to by the passed address and returns a status integer to the calling routine. A status = 0 indicates overall success; any other return status indicates an error condition.

This function performs the following tasks: 1) resizes the *Mission Image*, 2) scales and computes input image magnitudes, 3) Performs a prewarp on the magnitude mission image, 4) computes a dense set of image control points, and 5) resamples the mission image according to the just computed control points. We will describe the first two tasks here, leaving a more complete description of tasks (3), (4) and (5) for the next sections. Many, but not all, of the signal processing functions can be efficiently parallelized. MREG source code includes OPENMP pragmas to produce multi-threaded executables whenever the appropriate compiler directive is set.

Since MREG does not modify the *Reference Image* in any way, the assumption is made that the *Registered Image* produced is made identical in dimensions to the *Reference*. The *Mission Image* is not constrained to be of those dimensions. Hence, the first task of *mregister*() is to resize the *Mission Image* to that of the *Reference*. This is done in one of two ways depending on whether or not the input images are geocoded. If the inputs are not geocoded (e.g. are in radar image coordinates; Geo_Flag = "N"), then the *Mission* image is truncated or zero-filled, whichever is necessary, in order to make its dimensions equal to those of the *Reference*. This is done in such a way that the GRP pixels (Img_GRP) are coincident, so some amount of translation may be accomplished in this step as well. If the input images are geocoded (Geo_Flag = "Y"), then the GRP pixel coordinates are ignored and the corner coordinates in latitude and longitude are used instead. The *Mission* image is resized such that it corresponds to the geographical bounding box of the *Reference*. This section of the code is multi-threaded and uses no additional memory (utilizing the allocated, but as-yet unused, *Registered* array as scratch space).

The next step is to detect and scale the input images. The registration algorithms of MREG do not rely on the complex phase of the images. This, in fact, is what makes it a non-coherent registration processor suitable for tasks such as stereoscopy and Two-Color Multiview. Instead, both input images are magnitude detected. During the course of this process, they are also scaled to have the same rms value and to be efficiently represented by 16-bit integer pixel values. The 16-bit magnitude images are both stored in the *Registered* image array space, so again no additional memory is allocated for this step. The scaling / detection routine is multi-threaded.

3.2.1 Prewarp

The third step is to perform a prewarp on the just-computed magnitude detected *Mission* image. The purpose of prewarping is to remove known deterministic offsets from the input image pair that can be determined from the imaging geometry metadata, thus potentially saving a considerable amount of computation in later stages. Note that for geocoded images, this step is bypassed since geocoding by its nature produces “nearly” registered images. Some factors that are taken into account in the prewarp stage are differential rotations, scale factors, and different image planes. Note that ground plane *displacements* are handled in the resizing step. The prewarp is accomplished with an affine transformation, whose 2X2 transformation matrix may be defined as:

$$\boldsymbol{Q = M_2^{-1}\, P_2\, R\, P_1^{-1}\, M_1} \tag{1}$$

where the subscripts refer to the *Reference Image* (1) or *Mission Image* (2), the M matrices scale from pixels to meters, the P matrices are ground plane to image plane projections, and R is a ground plane rotation. (Straight line projections are used by the P matrices for expediency; more precise registration will follow.) What is desired is an overall mapping from image plane pixels (*Reference*) to image plane pixels (*Mission*). This is accomplished in the five steps above (pixels to meters, image plane to ground plane, rotation in ground plane, ground plane to image plane, and finally meters to pixels. If we denote

$$\boldsymbol{a = (\hat{x}_i \cdot \hat{x}_f)/ss_x} \tag{2}$$

$$\boldsymbol{b = (\hat{x}_i \cdot \hat{y}_f)/ss_x} \tag{3}$$

$$\boldsymbol{d = (\hat{y}_i \cdot \hat{y}_f)/ss_y} \tag{4}$$

In these expressions, the $\hat{x}$ and $\hat{y}$ refer to the cross range (x) and range (y) image axis unit vectors, the subscripts refer to the image plane (i) and the focus plane (f), and the *ss* values are the x and y image scale factors (m/pix). With these substitutions and combining the M and P matrices, it can be shown that the affine matrix Q is:

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{a} & \boldsymbol{b} \\ \boldsymbol{0} & \boldsymbol{d} \end{bmatrix}_2 \begin{bmatrix} \boldsymbol{\cos\theta} & \boldsymbol{-\sin\theta} \\ \boldsymbol{\sin\theta} & \boldsymbol{\cos\theta} \end{bmatrix} \begin{bmatrix} \boldsymbol{1/a} & \boldsymbol{-b/ad} \\ \boldsymbol{0} & \boldsymbol{1/d} \end{bmatrix}_1 \tag{5}$$

Here, the matrix subscripts denote the fact that the component coefficients are computed from the *Reference* metadata (1) or the *Mission* metadata (2) using the equations above. The angle θ is the ground plane rotation angle from *Reference* to *Mission*. Thus, the matrix Q maps pixel locations in the *Reference* image to corresponding pixel locations in the *Mission* image. This, of course, is an approximate mapping relying only on the image metadata. However, it does account for differing image azimuths, depression angles, image planes, and scale factors. The matrix Q is computed in a routine called *find_prewarp*(), which returns the four coefficients of Q in the arrays *ax*[] and *ay*[].

What remains is to perform the actual warping. This is accomplished in the routine *prewarp*() using a conventional bilinear interpolator. The code is multi-threaded and allocates an additional 16-bit image array as scratch space. Note that it is the *detected Mission* image that is resampled, not the complex *Mission* image. This scratch array represents the single largest memory use besides the three complex images, thus establishing the overall memory requirement of MREG at 3.25 times that of the *Reference* image.

3.2.2 Control Point Computation

After prewarping, the routine *compute_cp*() is called next by *mregister*. This routine is the heart of the multi-scale registration algorithm. Starting with the detected *Reference* image and the resized, detected, and prewarped *Mission* image (hereafter called the *Source* image), its function is to compute a very dense set of *control points* or local measurements of image-to-image displacements over the extent of the input images. These displacements are stored in two arrays, *cpxd*[] and *cpyd*[], holding the 2-D x- and y- displacement surfaces respectively.

The methodology used is a classic multi-scale approach. The algorithm loops through typically 6 stages starting from the lowest resolution and ending at full resolution. At each stage, the detected images are low pass filtered and downsampled to the stage resolution. The filtered and downsampled images are then compared at many local regions, the displacement vector at each region is computed. These displacement vectors are found by minimizing a *mean squared distance* metric, not a cross-correlation. While both msd- and correlation-based metrics are suitable for coherent pairs, the msd approach is much more effective for non-coherent image pairs. At the next stage, these local displacements are upsampled to the new (higher) stage resolution and the process is repeated. The displacements are thus accumulated from stage to stage, refining the overall displacement surfaces until they are found at the last, full resolution, stage. Since the resolution stages must be sequential, the code multi-threading is accomplished within the stage loop by spreading the local region computations across the threads.

This critical component of MREG is controlled by a set of 5 parameters, passed to it in the *Pargs* structure which in turn is defined in *mreg.h*. These parameters may be optimized for specific applications as discussed in the User Guide, but it may be noted that for a great majority of tasks, a default set has proven to be quite reliable. We will first discuss these parameters.

As has already been noted, the algorithm progresses in resolution stages. The integer *nstages* sets the total number of stages, and a default value of 6 is generally advisable. Thus, at the first stage, the images are filtered and downsampled by a factor of 2^5. Each subsequent stage has twice the resolution of its predecessor, with the final stage at full resolution.

The parameter *cpbox*[2] controls the size of the local region, in pixels, over which the local displacement vectors are computed. This is a two-valued array; the first value establishes the y (range) dimension of the local box and the second the x (cross-range) dimension. Unlike REGI, MREG does not allow the box size to vary from stage to stage. However, because of the downsampling, the effective box size in square meters on the ground varies with the stage resolution. The default values of *cpbox*[] are [31,31].

A third parameter, *cpspa*[], controls the spacing of the local boxes at each stage. Again, MREG does not allow for this parameter to vary among stages. This parameter determines the *density* of control points; image-to-image displacements for warping are subsequently interpolated from these control points to the full image resolution. The default values are [21,21] in the y and x directions, but some applications such as stereoscopy may call for much smaller values.

A fourth parameter controls the search size for finding local displacement vectors at the lowest resolution (first) stage. This parameter is called *cpsrch*[2]. The default values are [2,2], but are sometimes set significantly larger than this. A search value of k means that the local displacement vector will be exhaustively searched in that dimension over +/- k pixels at the first stage. This corresponds to a full-resolution search distance of $(2k + 1)2^{n}$ pixels, where n is the downsample factor for the first stage. Using default values, this is equivalent to a total local 2-D search area of 160 x 160 = 25,600 full resolution pixels, remembering that deterministic offsets have already been removed in the prewarp. While this is adequate for most situations, for large image sizes and significant terrain-induced displacements, this parameter must be substantially increased at the cost of computation time.

The final parameter, *cpfiltbox*[2], is rarely modified from its default value of [7,7]. This parameter controls a 2-D median filter that is applied to the computed displacement surfaces at each stage to remove spurious local measurements.

Having introduced the control parameters, we now present a summary of the multi-stage control point computation. It will be seen that image registration by the multi-scale approach is an exercise not so much in signal processing as it is in bookkeeping. That is, the actual computations required are relatively trivial, but keeping track of all of the loop variables and pixel indices is quite exacting. Thus, this section is more descriptive than mathematical.

At the outset, we will note that, while numerous scratch arrays are allocated from memory, they are individually quite small compared to that required for the input complex images. Collectively, they are also small compared to the scratch array allocated by *prewarp*(), so the total memory requirement discussed earlier stands. The multi-threading is accomplished perforce *within* the multi-scale loop, so all memory allocation is performed but once outside of that loop.

Except for the aforementioned memory allocations, all of the code in *compute_cp*() occurs within a resolution stage loop. The first iteration of the loop uses a decimation factor of $deci = 2^{(nstages-1)}$, and the factor is reduced by 2 for each subsequent stage. All stages use the same values for *cpbox*[], *cpspa*[]. The search size starts with *cpsrch*[] for the first stage, but the value is decreased by a factor of 2 for each subsequent stage. This parameter is bounded from below by 1 for all stages except the last (full resolution) for which it is bounded from below by 2. Thus, at least a +/- 2 pixel search in both dimensions is performed at full resolution.

The first step in the stage loop is to upsample previously determined 2-D displacement surfaces to the new resolution. The prior displacement surfaces are stored in the arrays *cpxdp*[] and *cpydp*[] and are upsampled into the arrays *cpxd*[] and *cpyd*[]. The algorithm also keeps track of regions outside the domain of legitimate image data, as for example the zero-filled portion of the resized or prewarped *Mission* image. These areas are denoted by a NULL value of -9999 in the surface arrays. In the first (lowest resolution) loop stage, *cpxd*[] and *cpyd*[] are initialized to the NULL value. This step is not multi-threaded.

The second step is to filter and downsample the detected *Reference* and the resized, prewarped, and detected *Source* image by the factor *deci*. The filter used is a boxcar filter of size (deci+1) in each dimension. This routine is normalized to unity gain and is multi-threaded.

Next, we enter a nested pair of loops over the set of control points (in the y- and x- directions). These extensive loops are multi-threaded. Patches of the *Reference* and *Source* images, centered at each control point location, are extracted from the corresponding detected arrays, type converted to floats, and *equalized*, i.e. the reference patch values are offset by a constant such that both patches have the same rms value.

We then enter a pair of even deeper loops (within the control point loops) over the y- and x- search distances. For each y- and x- control point (outer two loops), and each y- and x- search space value (inner two loops), we find the "best" displacement vector. The search space is centered not on the control point location, but rather that location summed with the (previous stage's) accumulated displacement surfaces *cpxd*[] and *cpyd*[] at that location. For each search value, a *cpbox*[] sized patch is again extracted from the *Source* image and compared to the previously extracted, equalized *Reference* patch. The comparison is made by computing the mean squared difference:

$$\boldsymbol{msd_{xy} = \frac{1}{mn}\sum_{m,n}\left(pix_{source} - pix_{reference}\right)^2} \quad \textbf{(6)}$$

where the sum is over the [m,n] size of the image patches. The value for *msd* is computed for every x- and y- search space value in the inner loops, identifying the global minimum. For all stages except the last, the integer values of the x- and y- search values yielding the global minimum mean squared distance are then summed to the displacement surface accumulators *cpxd*[] and *cpyd*[]. For the last stage, the two-dimensional array of search space msd values are interpolated to sub-pixel accuracy by a 3x3 point interpolator before being summed to the accumulators. Reviewing the above, it will be seen that we determine the minimum msd over an exhaustive search space given by:

$$\forall_{\boldsymbol{tp\_x}}\forall_{\boldsymbol{tp\_y}}\, \mathbf{min}_{(x|y)}\left(\boldsymbol{msd_{xy}}\right) \quad \textbf{(7)}$$

That is, *for each* control point location in x- and y-, we find the minimum msd over the 2D search space interval $y \in$ [-cpsrch[0],+cpsrch[0]], $x \in$ [-cpsrch[1],+cpsrch[1]] centered on the (offset) control point location.

After the outer (control point) loops are exhausted, the thus-far accumulated displacement surfaces *cpxd*[] and *cpyd*[] are median filtered in the routine *medfilter*(). The filter size is determined by the parameter *cpfiltbox*[2]. This quite standard function is accomplished by a call to the *Numerical Recipes* routine *nr_select*(). Finally, after the last resolution stage is completed, *compute_cp*() exits back to *mregister*(), passing the accumulated displacement surfaces.

3.2.3 Complex Image Warp

Having found the two-dimensional displacement surfaces *cpxd*[] and *cpyd*[], all that remains is to resample the original complex *Mission* image to overlay the *Reference* image. This output *Registered* image array has already been allocated (and used as scratch memory at various steps).

This resampling step must take into account both the deterministic affine transformation used in the prewarp as well as the non-deterministic two-dimensional displacement surfaces found by *compute_cp*(). Furthermore, the displacement surfaces were determined on the control point grid, which is coarser than full-resolution by the *cpspa*[] box spacing, thus necessitating an interpolation to full resolution. Note that even though the final warp encompasses both the deterministic and non-deterministic components, only *one* complex resampling is performed on the *Mission* image, rather than two successive warps. This is done to minimize the frequency space scalloping that, however minimized, is an unavoidable consequence of any resampling filter.

The complex image warp is accomplished in the routine *cwarp*() which in turn calls the routine *spline_warp_c*(). *cwarp*() performs the "bookkeeping" and *spline_warp_c*() performs the actual 4x4 point complex-valued spline interpolation. The code is multi-threaded at the outermost loop and requires essentially no additional memory. Looping on the samples of the output image array (denoted (i,j)), *cwarp*() first finds the four nearest neighbors from the control point displacement surfaces *cpxd*[] and *cpyd*[]. These displacement values are bilinearly interpolated to the output sample location (i,j). The contribution from the prewarp affine transformation, Q, at this location is then summed in. At this point, we have computed the full-resolution, floating point location in the *Mission* image (denoted (i',j')) corresponding to (i,j). Generally speaking, this *Mission* image location (i',j') does not fall on a discrete sample value (i.e. the floating point values of i' and j' are not integers). The interpolator in *spline_warp_c*() then performs a 4x4 point Catmull-Rom cubic spline interpolation in the neighborhood of (i',j') to determine the complex value of the *Mission* image at (i',j'). This value is placed into the output *Registered* image array at location (i,j). The interpolator has a (1D) kernel given by [(Glassner 1995)]:

$$\boldsymbol{h(x)} = \begin{bmatrix} \mathbf{1 - 2.5|x|^2 + 1.5|x|^3} & \mathbf{0 < |x| < 1} \\ \mathbf{2 - 4|x| + 2.5|x|^2 - 0.5|x|^3} & \mathbf{1 < |x| < 2} \\ \mathbf{0} & \boldsymbol{otherwise} \end{bmatrix} \qquad \mathbf{(8)}$$

3.2.4 Outputs

The primary output of *mregister*() is, of course, the *Registered* complex image with dimensions exactly equal to those of the *Reference* image. Two other outputs are optionally available. These are the actual x- and y- displacement surfaces. These outputs are critically important in applications of stereoscopy since they provide the subsequent stereo mapping algorithms with the complete 2-D correspondence map. This includes both the deterministic as well as the terrain-induced components. Therefore, these two output files, *xmap.srf* and *ymap.srf* contain important metadata fields for use by subsequent functions. The output of these arrays is handled by the function *write_xy*().

4. MREG USER'S MANUAL

The processing algorithms of MREG may be executed from the command line via the *mreg()* wrapper. Conversely, they may be executed in an embedded application via the *mregister()* routine. Either way, the behavior of the software and its outputs are identical. In this section, we will focus on the command line environment. However, suggestions on selection of runtime parameters apply equally to the embedded situation.

4.1 The MREG command line

MREG is executed from the command line. A built-in help menu may be accessed by typing:

```
%mreg
```

In order to register an image pair, MREG requires two input image arguments and accepts several options. The two input images must be in .SRF format and complex, i.e. image pixels are represented by I- and Q- values, either floating point (C8) or short integers (IQ4). If we denote these two images as *image1.srf* and *image2.srf,* then the simplest possible command is:

```
%mreg image1.srf image2.srf
```

The first image (i.e. *image1.srf*) will be treated as the *Reference Image* and the second as the *Mission Image.* Execution will result in three output files, the complex *Registered Image, reg.srf,* and the two displacement surface images, *xmap.srf* and *ymap.srf.* These two are in real, floating point (R4) format.

4.2 Command line arguments

With no other command line parameters specified, the above example will result in the registration algorithms employing the default runtime parameters. These default parameters may be determined by accessing the help menu. The operation of MREG may be altered through the use of command line options. Options are invoked by a dash (-) followed by an option keyword and argument. Only the first letter, or specified critical letter, of any keyword need be typed. Options may be specified in any order, but its corresponding argument must follow immediately. The options supported are:

-correlation_box_size n This option specifies the values for *cpbox[]*. While the software supports different row and column values for *cpbox[],* the command line wrapper is confined to equal row and column values of **n**. Values for **n** must be odd. Values exceeding 255 might be considered imprudent. See discussion below.

-s(p)acing_size n This option specifies the values for the correlation box spacing, *cpspa[]*. Again, the command line wrapper confines the choice to equal row and column values of **n**, although the software supports unequal values. This spacing is generally 2/3 of the correlation box size. The values of *cpspa[]* also determine the downsample ratio applied to the x- and y- displacement surface files.

-search_size n The search distance parameters, *cpsrch[]*, can be specified by this option. Again, the command line wrapper only supports equal row and column values. As discussed in Section 3.2.2, the specified search size applies only to the first iteration of the control point computation. Subsequent iterations are computed as discussed therein. These values have a very large impact on computation time and, of course, search distance.

-filter_size n The median filter size parameters, *cpfiltbox[]*, are rarely changed. The value for **n** must be odd and is constrained by the command line wrapper to be equal in row and column values.

-number_stages n This value specifies the number of multi-resolution stages, *nstages*. As such, it controls a great many aspects of the control point computation process.

4.3 Discussion

The default values of the runtime parameters have been carefully chosen and will result in excellent performance in many situations. However, there are certain applications that may call for somewhat different choices of runtime parameters for optimum performance. We will discuss these situations here in a general way. Any given application may call for some fine tuning of these parameters beyond these general considerations.

4.3.1 CCD

Generally speaking, CCD applications require that the images to be registered must be imaged using a fairly strict set of geometrical constraints on differential depression angles and azimuth angles (see (Jakowatz 1996)). However, CCD may be pursued using quite different squint angles (ground track angles) for the pair. Thus, we have two situations. For a parallel ground track image pair geometry, the closely similar depression, azimuth, and squint angles limits the total amounts of differential layover and rotation in the image pair to rather benign values, even where substantial amounts of terrain relief are present. In this case, the default runtime parameter values should prove adequate. In fact, where highly accurate geometry repeats are engineered into the system, it may be possible to improve computational throughput by reducing the number of stages and/or increasing the box spacing over the default values.

On the other hand, substantially differing ground tracks present a formidable registration problem in large terrain relief areas due to differential layover. This is similar to the stereo registration problem discussed below and has the same solution. For these situations, we require the registration algorithm to a) search over a larger displacement space, and b) produce a finer grid of control points in order to accommodate the spatially-varying layover component. Should the default parameter set prove inadequate for this situation, it is recommended (in order of preference) to reduce the spacing size, increase the search size, decrease the correlation box size, and increase the number of stages. It is unlikely that all of these changes would prove necessary except in the most extreme situations, but the first two are rather more likely.

4.3.2 TwoColor MultiView

TwoColor MultiView (2CMV) is rather like CCD except the image pair need not be phase coherent. This lack of a coherency requirement means that the geometrical imaging constraints placed on the collection of such a pair of images can be significantly relaxed over that imposed for CCD. This exacerbates the registration effort required. Relaxed constraints on differential depression, squint, and azimuth angles all lead to more rotation, scale change, and differential layover effects in the image pair. With accurate metadata available, the prewarp stage of MREG can accommodate much of the registration discrepancy attributed to deterministic sources of rotation, translation, and change of scale. However, differential layover is terrain and geometry dependent and therefore cannot be deterministically removed via the prewarp. (In passing, we note that the use of a terrain model input source can further eliminate much of the terrain induced spatially variant layover. This capability has been developed for the REGI processing environment. It may be introduced into the MREG environment at some future time.) The default parameter set has proven to be adequate for most 2CMV applications to date, but some situations may call for a finer control point grid (smaller box spacing size) and a larger search space (increased search size and stages).

4.3.3 Stereoscopy

SAR stereoscopy makes deliberate use of height dependent layover in order to deduce 3-D terrain [(Eichel 2002)]. The requisite registration process must therefore accommodate significant differential layover in the range, cross-range, or both dimensions. Because the collection geometries are chosen with substantial differences in grazing, azimuth, and squint angles, this application is arguably the most demanding on the registration process. Consequently, stereoscopy will often require more liberal (and time consuming) choices in runtime parameters. Typical stereoscopy applications in Dept. 5962 have used a correlation box size of 17, search size of 11, spacing of 7 or 9, and 6 or 7 stages. The very dense control point spacing is required to generate high resolution x- and y-displacement maps for subsequent processing into 3-D data. Note that for SAR images covering many square kilometers, large differences in grazing or azimuth angles, and significant terrain relief, spatially dependent differential layover of hundreds or even thousands of pixels is not unusual. Such extreme situations may be accommodated by increasing the search distance and number of stages accordingly.

5. CONCLUSIONS

In this report we have traced the evolution of a series of increasingly powerful SAR image registration algorithms that have played a critical role in the development of extremely useful applications such as two-pass interferometry, CCD, TwoColor MultiView, and SAR stereoscopy. This evolution culminated in the SNL REGI processing suite, circa 2005. While extraordinarily capable, that software has not been well documented and is intractable for embedded applications. This report describes in detail the subsequent MREG software that, while based on the same processing algorithms, is at once more transparent, more accessible, and eminently embeddable. We hope this document sheds vital clarification on this extremely robust SAR image registration technique.

6. REFERENCES

Eichel, P.H. "Fast algorithms for 3-D terrain mapping using Spotlight-Mode SAR stereo." *Proc. for the Workshop on Synthetic Aperture Radar Technology*. Redstone Arsenal: U.S. Army Aviation & Missle Command, 2002.

Eichel, P.H., C.V. Jakowatz, Jr., and P.A. Thompson. "Research Report No. 93/12/16: Interferometric processing at Sandia National Laboratories, 1991-1992." SNL 5900 Classified Report, Albuquerque, 1993.

Gabriel, A.K. and R.M. Goldstein. "Crossed orbit interferometery: theory and experimental results from SIR-B." *Int. Journal Remote Sensing, Vol. 9, no. 5*, 1988: pp. 857-872.

Glassner, A.S. *Principles of Digital Image Synthesis.* San Francisco: Morgan Kaufmann Publishers, Inc., 1995.

Graham, L.C. "Synthetic interferometer radar for topographic mapping." *Proc. IEEE, Vol. 62, no. 6*, 1974: pp. 763-768.

Jakowatz, C.V., D.E. Wahl, P.H. Eichel, D.C. Ghiglia, and P.A. Thompson. *Spotlight-Mode Synthetic Aperture Radar: A Signal Processing Approach.* Boston: Kluwer Academic Publishers, 1996.

Press, W.H., and S.A. Teukolsky. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge: Cambridge University Press, 1992.

DISTRIBUTION

1	MS0519	J.G. Chow	05349
1	MS0519	R.D. West	05346
1	MS0519	D.L. Bickel	05344
1	MS0533	R.R. Riley	05342
1	MS1207	T.M. Calloway	05962
1	MS1207	C.V. Jakowatz, Jr.	05962
1	MS1207	N.E. Doren	05962
1	MS1207	I.A. Erteza	05962
1	MS1207	D.E. Wahl	05962
1	MS1207	D.A. Yocky	05962
10	MS1207	P. H. Eichel	05962
1	MS0899	Technical Library	9536 (electronic copy)

Sandia National Laboratories