G2 SAR Processor (Ver. 1.0) User Manual

J.M. Horrell, R.T. Lord and M.R. Inggs

February 13, 2001

Radar Remote Sensing Group
Department of Electrical Engineering, UCT
Private Bag, Rondebosch 7701, South Africa

Contents

1 Scope

This report is a user manual describing the G2 *Synthetic Aperture Radar* (SAR) processor (Ver. 1.0), and it was written by the *Radar Remote Sensing Group* (RRSG) at the University of Cape Town.

2 Introduction

The G2 SAR processor is based on the range-Doppler algorithm and has been designed for the processing of airborne SAR data. The processor is modular and flexible and can handle a wide range of SAR processing tasks. This version of the processor includes a Python program, "`g2.py`", which integrates the various modules and has been designed for the processing of SASAR VHF data (South African SAR VHF sensor). In many cases, it is possible to use the integrated version for processing of data from systems other than the SASAR VHF system by simply changing input parameters. Where the integrated processor is found to be unsuitable for a particular processing job, it might still be possible to configure the various modules manually to get the job done (see Section 8). In addition, if semi-automated processing is required for a new SAR sensor, a new version of the overall glue program in Python could be created which uses the same core modules (such as range compression, azimuth compression, etc.).

3 Quickstart

For a quickstart to the integrated processor, type the following from the command line:

```
g2.py --help
```

or simply:

```
g2.py
```

Both of the above commands will display the help screen shown below, which describes the syntax required for operation, how to generate templates of the required processor- and radar configuration files, and also how to generate an HTML user manual:

```
Quick help screen:
------------------

RUN PROCESSOR: (perform SAR processing - see "create template files" be-
low)
usage:  g2.py [processor_config_file_name]

CREATE TEMPLATE FILES: (create template processor- and radar config files)
usage:  g2.py --templates

USER MANUAL: (install HTML user manual in current dir)
usage:  g2.py --user-manual

QUICK HELP: (this screen)
usage:  g2.py --help
```

Note: To update the HTML user manual, copy any HTML source changes into the “`createHTMLUserManual()`” function in the file “`g2tmpl.py`”. This will allow users of the processor to automatically generate the latest version of this document in their current directory.

Therefore, to process a SAR image, the processor operator would type:

```
g2.py processor_config_file_name
```

The reader is referred to Section 9, which works through a typical SAR processing example in a step-by-step manner.

4 Configuration Files

The G2 processor in its integrated form relies heavily on two ASCII configuration files being properly set up, namely the *processor configuration file* and the *radar configuration file.* These files contain parameter names and values and must be configured prior to a processing run. The processor will automatically generate templates of these two configuration files in the correct format by typing:

```
g2.py --templates
```

The two generated configuration files are listed in Appendix A and B. Note that they include help on parameter usage.

Below is a short description of the two configuration files:

Processor Configuration File — This file is configured prior to processing by the processor operator and contains all those parameters which the processor operator usually would select (like the required azimuth resolution). For normal operation, the processor configuration file would be the *only* file with which the processor operator needs to be concerned.

Radar Configuration File — This file is ideally written by the radar control software and would not normally be altered by the processor operator. Parameters in the radar configuration file include radar centre frequency and pulse length, for example. The radar configuration file also contains names (without paths) of the raw data file, the LBR file, the DGPS file, etc. The paths to these files are set up in the processor configuration file.

5 Input and Output Data Formats

5.1 Input Data Format

The format of the raw SAR data is specified in the radar configuration file (see Appendix B, the `$RawDataType` variable), and can be:

- byte (8-bit I, 8-bit Q, unsigned char)
- float (32-bit IEEE float I and Q, little endian)

Since the data is expected to be in complex format (I and Q), every range sample in byte-format is 16-bits long, and every range sample in float-format is 64-bits long.

5.2 Output Data Format

The output data format of the integrated processor (`g2.py`) is specified in the processor configuration file (see Appendix A, the `$DetectMethod` variable), and can be:

- complex float (2 x 32-bit IEEE float I and Q)
- real float (1 x 32-bit IEEE float I and Q)

Note that the endian type can be specified with the `$OutputEndian` variable.

Setting the detect-method to `cmplx` will result in a *single look complex* (SLC) image, and requires that the number of azimuth looks is set to 1.

Setting the detect-method to either `mag`, `pow` or `powdB` will result in a real image (in float format) that has been obtained by taking either the magnitude, power or dB power of the complex data.

Note that using the *azcom* module in standalone mode (see Section 8) allows one to set the *real* output data type to either:

- unsigned char (1 byte)
- unsigned short int (2 bytes)
- long int (4 bytes)
- float (4 bytes)
- double (8 bytes)

6 Design Philosophy and Flexibility

The processor has been written to be as flexible as possible (required for experimental systems) whilst retaining a simple operator interface. A very modular approach has been taken with modules usually reading from some input file and writing to an output file. Log files are also generated by certain of the modules. These are read by later modules to set parameters in the integrated processor.

The operator can, through the processor configuration file, decide whether to keep all the intermediate temporary files and temporary log files, or to delete them at the end. For debugging a processing run, it is often very useful to have all the temporary log- and config (and even data) files available, but these can take up a lot of disk space.

Most of the modules are written in C (for speed) with certain modules being written in the Python language. The overall glue for the processor is written in Python. The C modules have been compiled to be standalone executables, and the Python modules have been compiled to Python byte code which are called by the main Python code. The Python code automatically configures the processing for the particular module.

For example, the azimuth processing module requires a configuration ASCII text file which the Python code automatically generates before calling the operating system to excute azimuth processing. As already mentioned, this approach also allows the flexibility of running the C executables independently from the rest of the processor.

7 Integrated Processor Configuration

For basic operation, see the Quick Start section. In the integrated processor (`g2.py`), modules may be switched in and out (y/n/only/off states are possible - see the notes in the processor configuration file in Appendix A for details), which allows for very flexible operation. Using this approach, it is possible to proceed half-way through a processing run, stop and examine the intermediate files, and then continue where the processor left off. Another use might be to omit modules altogether which are not relevant for the particular radar.

Section 8 describes the standalone modules called by the `g2.py` program. For more detail on the integrated processor operation, a quick scan through the `g2.py` file will give the user a good idea of the top-level operation. The `g2.py` file is well commented and should be fairly readable even to those unfamiliar with the Python programming language (note that in Python, logical grouping is done by indentation, not brackets).

8 Standalone Module Configuration

As an alternative to the integrated processor approach, the major modules can also be run from the command line, independently from the rest of the processor. This is possible as these modules are in fact standalone executable programs. As for the integrated processor (`g2.py`), some of the major modules also depend on their own ASCII configuration files being set up. The ability to run the major modules separately from the command line significantly adds to the flexibility available to the user, extending the processor's usefulness well beyond the SASAR VHF system.

The modules which may be run in standalone mode are (in normal order of SASAR processing execution):

- **imu_unpack** — (C executable - SASAR specific). Unpacks the binary format LBR motion records to an ASCII file. Type "`imu_unpack`" for usage.
- **g2unpk_dgps.py** — (Python script - SASAR specific). Parses DGPS file from OmniStar system and extracts relevant part to ASCII file for processing. Type "`g2unpk_dgps.py`" or "`python g2unpk_dgps.pyc`" for usage.
- **g2mocfilt.py** — (Python script - slightly SASAR specific - rewriting this in C will speed things up). Merges the data streams from the IMU and DGPS (including smoothing, interpolation, etc.). This module takes as input the two output ASCII files from the previous two steps. The output is a single ASCII file with the merged latitude, longitude and altitude data. Type "`g2mocfilt.py`" or "`python g2mocfilt.pyc`" for usage.
- **mocomp** — (C executable - slightly SASAR specific). Calculates the range shift required for each pulse of the processing run. This takes as input the output merged data file from the previous step. The output is in the form of a binary file containing pulse numbers (4-byte integers) and range shifts (4-byte floats). A log file may be generated which contains the average ground speed calculated from the motion data for the scene to be processed. Type "`mocomp`" for usage.
- **sniffdc** — (C executable). Finds the DC offsets and average I/Q channel ratio for the scene. The values calculated may be written to a log file. Type "`sniffdc`" for usage.
- **rngcom** — (C executable). Motion compensation correction, interference suppression and range compression. Type "`rngcom`" for usage. The automatically generated template rngcom configuration file contains notes on setting up parameters.
- **stepf** — (C executable). Performs stepped frequency processing. Combines the returns from a stepped frequency burst of pulses into a single range line (with combined bandwidth, upsampled, etc.). Type "`stepf`" for usage.
- **corner** — (C executable). Corner turns (transposes) data. Reads from a data file, corner turns and writes out again to a new file. Type "`corner`" for usage.
- **azcom** — (C executable). Performs range curvature correction, azimuth compression and multilook processing. Reads the corner turned file and writes out the azimuth compressed file (image). Type "azcom" for usage.
- **iq2mag** — (C executable). Converts complex IQ data to magnitude/power. Reads and writes to disk. Type "`iq2mag`" for usage.
- **flt2byte** — (C executable). Converts floating point binary data to unsigned char binary data (scaled). Type "`flt2byte`" for usage.
- **b2tif** — (C executable). Writes unsigned char binary data (image) to TIFF format. Type "`b2tif`" for usage.

- **swapend** — (C executable). Swaps endian format of binary data on disk. This is useful if moving data from i386 (little-endian) to Sun (big-endian), for example. Type “`swapend`” for usage.

9 Processing Example

This section will describe in a step-by-step manner the entire process that needs to be carried out in order to process a raw SAR image.

First, we will need to create the directories that are going to hold both the input and the output data. For the input data:

```
mkdir -p /data/19990902
```

It is advisable to work out a naming convention for the directories; here we have chosen the date at which the data was recorded as the name of the input-data directory. This directory contains the following files:

- Raw SAR image (say `CAL5-H3.000`)
- LBR file (say `CAL5.001`)
- DGPS file (say `CAL5.GPS`)
- Radar Configuration File (say `cal5-vv-radar.cfg`)

The radar configuration file will have to be modified by the processor operator (after having created a template as described in Section 4), and might look something like this:

```
SASAR VHF configuration file for G2 Processor
$RadarConfigVersion => 0.4
=============================================
$DataID                                   => 19990902-CAL5-000-VV
$RawDataFile (no path)                    => CAL5-H3.000
$LBRFile (no path)                        => CAL5.001
$DGPSFile (no path)                       => CAL5.GPS
$RawDataType (byte/float - see note)      => byte
$DataStartLP (LBR PRI for raw data start) => 5440
$RadarDelayToStartSample (secs)           => 11.50e-06
$RadarPRF (as in raw data file - Hz)      => 136.363636
$RadarAzSamples (in raw data file)        => 24880
$RadarRngBins (in raw data file)          => 4096
$RadarA2DFreq (Hz)                        => 24.0e+06
$RadarCarrierFreq (nominal - Hz)          => 141.0e+06
$RadarPulseLength (sec)                   => 10.5e-06
$RadarChirpBandwidth (Hz - zero for mono) => 12.0e+06
$RadarMocTimeOffset (sec)                 => 0.5
$TerrainAlt (m)                           => 850

--Stepped Freq Setup (see note) --
$StepFreqMode (no/normal/user)            => no
$NumberOfFreqSteps (normal)               => 1
$FirstStepCentreFreq (normal - Hz)        => 141.0e+06
$StepFreqStepSize (normal - Hz)           => 12.0e+06
```

```
$StepFreqUserFile (user - no path)        => null

--Optional Params (else 'null' - see note)--
$DCOffsetI                                => null
$DCOffsetQ                                => null
$IQRatio                                  => null
$AveGroundSpeed (m/s)                     => null
```

Now let us create the directories holding the output data files:

```
mkdir -p /data/upington/config
mkdir -p /data/upington/images
```

Note that the word “upington” describes the directory contents more clearly than “19990902”. The *config* directory will contain the processor configuration files (often 4 per data take - one for each polarisation), and perhaps scripts to run the G2 processor in batch mode, for example:

- `cal5-vv-proc.cfg`
- `runall` (script to run the above configuration file(s))

The *images* directory will contain the processed image, and usually an image log file and a viewable `tif` image of the processed file, for example:

- `cal5-vv.pow`
- `cal5-vv.imlog`
- `cal5-vv.tif`

Before we can process the raw SAR image, the processor configuration file has to be set up (after creating a template as described in Section 4), and might look something like this:

```
G2 Processor config for SASAR VHF
$ProcConfigVersion => 0.7
=================================

---General Parameters---
$RunID (root of output file names)        => /data/upington/images/cal5-vv
$InputPath (location of raw data files)   => /data/19990902/
$RadarCfgFile (path optional)             => /data/19990902/cal5-vv-radar.cfg
$StartG2PRI (0 is start of raw data)      => 0
$InputPRIsToProc                          => 24880
$ProcPresumRatio                          => 5
$StartRngBinToProc                        => 360
$RngBinsToProc                            => 2300
```

```
---Proc Control [see note] ---
$EnableUnpackIMU (y/n/only/off)          => y
$EnableUnpackDGPS (y/n/only/off)         => y
$EnableMergeMocData (y/n/only/off)       => y
$EnableMocompCalc (y/n/only/off)         => y
$EnablePlotMotionError (y/n/only/off)    => y
$EnableSniffDC (y/n/only/off)            => y
$EnableRngProc (y/n/only)                => y
$EnableStepFreqProc (y/n/only/off)       => y
$EnableCornerTurn (y/n/only)             => y
$EnableAzProc (y/n/only)                 => y
$EnableFloat2Tiff (y/n/only/off)         => y
$EnableEndianSwap (y/n/only/off)         => y
$EnableOrient (y/n/only/off)             => y
$EnableImageLog (y/n/only/off)           => y
$EnableCleanUp (y/n/only/off)            => y

---RngProcStageParams---
$RngCompress (y/n)                       => y
$RngComRefPhase (+-1)                    => -1
$RngComWinConstTime [see note]           => 0.08
$RngComScale                             => 1.0
$MoComp (y/n)                            => y
$MoCompRngShiftFlg (y/n)                 => y
$RngShiftInterpSize                      => 8
$MoCompRngShiftSign [+-1]                => 1
$MoCompPhaseSign [+-1]                   => 1
$InterferenceSuppress (lms/notch/none)   => notch
$LmsUpdateRate                           => 1
$LmsNumWeights                           => 256
$LmsSidelobeOrder                        => 0
$NotchUpdateRate                         => 1000
$NotchNumFFTLines                        => 1000
$NotchCutoff (dB)                        => 2
$NotchMedianKernLen                      => 65

---StepFreqProcStageParams---
$StepFreqWinConstTime                    => 0.08

---AzProcStageParams---
$RngFocSegments                          => 256
$AzComRefPhase [+-1]                     => 1
$AzComNomAzRes [m]                       => 10.0
$AzComWinConstTime                       => 1.0
$AzComWinConstFreq                       => 0.08
$NumAzLooks                              => 1
$AzLookOverlapFrac [0.0-1.0]             => 0.5
$AzComRngCurvInterpSize                  => 8
```

```
$AzComInvFFTSizeReduc (power of 2)        => 2
$AzComRngCurvBatchSize                    => 256
$DetectMethod (cmplx/mag/pow/powdB)       => pow
$AzComScale                               => 1.0

---FloatToTiffStageParams---
$Float2ByteMath (none/powx/log/logpowx)   => pow0.25
$Float2ByteScale                          => 2.0

---EndianSwapStageParams---
$OutputEndian (little/big)                => little

---OutputOrientStageParams---
$OutputOrient (azline/rngline)            => azline
```

Now we are ready to process the raw image:

```
cd /data/upington/config
g2.py cal5-vv-proc.cfg
```

To view the image, we can type:

```
cd /data/upington/images
xv cal5-vv.tif &
```

It is unlikely that we will be 100% satisfied with the resulting image after the very first run. Very often the `$Float2ByteScale` scaling factor will have to be adjusted. However, there is no need to go through *all* the processing steps again. Simply adjust the follwing lines in the processor configuration file:

```
$EnableUnpackIMU (y/n/only/off)           => n
$EnableUnpackDGPS (y/n/only/off)          => n
$EnableMergeMocData (y/n/only/off)        => n
$EnableMocompCalc (y/n/only/off)          => n
$EnablePlotMotionError (y/n/only/off)     => n
$EnableSniffDC (y/n/only/off)             => n
$EnableRngProc (y/n/only)                 => n
$EnableStepFreqProc (y/n/only/off)        => n
$EnableCornerTurn (y/n/only)              => n
$EnableAzProc (y/n/only)                  => n
$EnableFloat2Tiff (y/n/only/off)          => y
$EnableEndianSwap (y/n/only/off)          => y
$EnableOrient (y/n/only/off)              => y
$EnableImageLog (y/n/only/off)            => y
$EnableCleanUp (y/n/only/off)             => y
```

and

```
    $Float2ByteScale                        => 3.0
```

Now run the G2 processor again:

```
    cd /data/upington/config
    g2.py cal5-proc.cfg
```

This iterative process will continue until we have obtained the best possible image. Note that we have used `xv` to view the image. Another popular image viewing program under Linux is `gimp`.

A Processor Configuration File

```
G2 Processor config for SASAR VHF
$ProcConfigVersion => 0.7
=================================

---General Parameters---
$RunID (root of output file names)        => ./run1-vv-1
$InputPath (location of raw data files)   => /scratch/sasar/
$RadarCfgFile (path optional)             => /scratch/sasar/run1-vv.cfg
$StartG2PRI (0 is start of raw data)      => 0
$InputPRIsToProc                          => 44560
$ProcPresumRatio                          => 5
$StartRngBinToProc                        => 0
$RngBinsToProc                            => 4096

---Proc Control [see note] ---
$EnableUnpackIMU (y/n/only/off)           => y
$EnableUnpackDGPS (y/n/only/off)          => y
$EnableMergeMocData (y/n/only/off)        => y
$EnableMocompCalc (y/n/only/off)          => y
$EnablePlotMotionError (y/n/only/off)     => y
$EnableSniffDC (y/n/only/off)             => y
$EnableRngProc (y/n/only)                 => y
$EnableStepFreqProc (y/n/only/off)        => y
$EnableCornerTurn (y/n/only)              => y
$EnableAzProc (y/n/only)                  => y
$EnableFloat2Tiff (y/n/only/off)          => y
$EnableEndianSwap (y/n/only/off)          => y
$EnableOrient (y/n/only/off)              => y
$EnableImageLog (y/n/only/off)            => y
$EnableCleanUp (y/n/only/off)             => y

---RngProcStageParams---
$RngCompress (y/n)                        => y
$RngComRefPhase (+-1)                     => -1
$RngComWinConstTime [see note]            => 0.08
$RngComScale                              => 1.0
$MoComp (y/n)                             => y
$MoCompRngShiftFlg (y/n)                  => y
$RngShiftInterpSize                       => 8
$MoCompRngShiftSign [+-1]                 => -1
$MoCompPhaseSign [+-1]                    => -1
$InterferenceSuppress (lms/notch/none)    => notch
$LmsUpdateRate                            => 1
$LmsNumWeights                            => 256
$LmsSidelobeOrder                         => 0
$NotchUpdateRate                          => 1000
$NotchNumFFTLines                         => 1000
```

```
$NotchCutoff (dB)                         => 2
$NotchMedianKernLen                       => 65

---StepFreqProcStageParams---
$StepFreqWinConstTime                     => 0.08

---AzProcStageParams---
$RngFocSegments                           => 256
$AzComRefPhase [+-1]                      => -1
$AzComNomAzRes [m]                        => 12.0
$AzComWinConstTime                        => 1.0
$AzComWinConstFreq                        => 0.08
$NumAzLooks                               => 4
$AzLookOverlapFrac [0.0-1.0]              => 0.5
$AzComRngCurvInterpSize                   => 4
$AzComInvFFTSizeReduc (power of 2)        => 2
$AzComRngCurvBatchSize                    => 256
$DetectMethod (cmplx/mag/pow/powdB)       => pow
$AzComScale                               => 1.0

---FloatToTiffStageParams---
$Float2ByteMath (none/powx/log/logpowx)   => none
$Float2ByteScale                          => 1.0e-5

---EndianSwapStageParams---
$OutputEndian (little/big)                => little

---OutputOrientStageParams---
$OutputOrient (azline/rngline)            => azline


NOTES:
=====
Proc Control:
    y    - Run this module, performing relevant parameter
           calculations or reads from log files, and write
           log file, if any.
    n    - Do not run this module, but infer parameters and
           read from log files as if it had been run.
    only - Run only this module.  Dynamic parameters are
           calculated or read from intermediate log files
           of the other modules unless these are set to "off"
    off  - Do not run this module and do not infer any
           parameters or read from its log file (i.e.  as if
           module absent altogrether).  Note that the ’off’
           state is only permitted for certain modules.  Checks
           are performed for modules not in ’off’ state which
           depend on a module which is in the ’off state.
    These options allow flexible processor configuration.  For
```

```
example, a run may be restarted from half-way, by setting
the previously completed portions to "n".  The correct
parameters for the later modules will still be calculated
or read from the intermediate log files.

The main modules (available as standalone modules, unless
                  contra-indicated):


UnpackIMU -
    Unpack the IMU records from the SASAR LBR file and write
    to an ASCII file (C executable).
UnpackDGPS -
    Unpack the DGPS records to a more readable ASCII format
    and sync with the IMU data (Python code).
MergeMocData -
    Merge the IMU and DGPS records to form a single ASCII
    file with the LBR PRI, Latitude, Longitude, etc.  This
    does all the smoothing, interpolation, etc.  (Python code
    which also requires the Scientific Python modules as
    freely available on the web).
MocompCalc -
    Calculate the range shifts required for each range line
    from the merged motion data.  Also creates the geocoding
    information.  (C executable).
PlotMotionError -
    Plot the motion compensation range shift as calculated
    by the MocompCalc module.  (Python code - not standalone)
SniffDC -
    Calculate the DC offsets and average I to Q value ratio
    from an analysis of part of the raw data (C executable)
RngProc -
    Range compression, interference suppression and motion
    compensation correction implementation (C executable).
StepFreqProc -
    Step frequency processing (C executable).
CornerTurn -
    Corner turn the range compressed file (C executable).
AzProc -
    Range curvature correction, azimuth compression and
    multilook (C executable).
Float2Tiff -
    Convert floating point output from azimuth compression
    to TIFF file for easy viewing (various C executables).
EndianSwap -
    Swap endian format (C executable).
Orient -
    Covert from azimuth line format to range line format
    (uses corner turn C executable)
ImageLog -
```

```
        Create image log file with geocoding info, etc.  (Python
        code - not standalone)
    CleanUp -
        Remove all temporary data and log files (Python code -
        not standalone).

RngComWinConstTime - set to 1.0 if step freq processing.
```

B Radar Configuration File

```
SASAR VHF configuration file for G2 Processor
$RadarConfigVersion => 0.4
=============================================
$DataID                                   => 19990721-tznn009-vv
$RawDataFile (no path)                    => tznn-h3.009
$LBRFile (no path)                        => tznn.009
$DGPSFile (no path)                       => tzaneen.gps
$RawDataType (byte/float - see note)      => byte
$DataStartLP (LBR PRI for raw data start) => 2415152
$RadarDelayToStartSample (secs)           => 10.6667e-06
$RadarPRF (as in raw data file - Hz)      => 136.363636
$RadarAzSamples (in raw data file)        => 44562
$RadarRngBins (in raw data file)          => 4096
$RadarA2DFreq (Hz)                        => 24.0e+06
$RadarCarrierFreq (nominal - Hz)          => 141.0e+06
$RadarPulseLength (sec)                   => 6.66667e-06
$RadarChirpBandwidth (Hz - zero for mono) => 12.0e+06
$RadarMocTimeOffset (sec)                 => 0.5
$TerrainAlt (m)                           => 1300

--Stepped Freq Setup (see note) --
$StepFreqMode (no/normal/user)            => no
$NumberOfFreqSteps (normal)               => 1
$FirstStepCentreFreq (normal - Hz)        => 141.0e+06
$StepFreqStepSize (normal - Hz)           => 12.0e+06
$StepFreqUserFile (user - no path)        => null

--Optional Params (else ’null’ - see note)--
$DCOffsetI                                => null
$DCOffsetQ                                => null
$IQRatio                                  => null
$AveGroundSpeed (m/s)                     => null

Notes
=====

RawDataType - byte (8-bit I, 8-bit Q, unsigned char)
              float (32-bit IEEE float I and Q, litte endian)

Stepped Freq Setup:
    For stepped freq operation, the "RadarPulseLen" and
    "RadarChirpBandwidth" parameters are taken to be those of each
    narrow band pulse (assumed constant for a run).  The step freq
    user file first line should be an integer which is the number
    of freq steps with the centre frequency of each step (in Hz)
    on subsequent lines.
```

```
Optional Params:
    These values are only used if the processor module where they
    are normally calculated is set to 'off'.  For example the DC
    offsets and IQ ratio are usually calculated in the SniffDC
    module.  The average ground speed is calculated in the motion
    compensation calculation module.
```