RITSAR

Douglas Macdonald

Rochester Institute of Technology

May 12, 2015

Contents

1 Introduction

Below is a description of each of the functions contained in the RITSAR package. For installation instructions, reference the README file.

Each of the sections below represent a module contained within the RITSAR package and each subsection details how to use each function. The format for the title of each subsection is "function(parameters)." This is followed by a short description of what the function does. Next, each parameter is defined. Finally, a description of what the function returns is given.

As an example, lets say you wanted to simulate your own phase history. The function required for this is simulate_phs located in the phsTools module. Before this function can be used, the required parameters need to be defined. The descriptions of each required parameter can be found in section 2.1. The first parameter that needs to be defined is "platform", which is a Python dictionary. A description of each key that goes into platform is also given in section 2.1. Once platform is defined, the user then creates an array specifying the locations of each point reflector in the scene as well as the corresponding amplitudes. After all this is done the user can then create their own phase history, stored in the phs variable, by typing the following into the Python console:

phs = ritsar.phsTools.simulate_phs(platform, points, amplitudes)

To get started, "sim_demo.py" in the ./examples folder can be run. This demo uses RITSAR's basic point simulator to generate a synthetic phase history. The default setup processes an X-band collect using the polar format algorithm. The final lines of main.py can be commented/un-commented to use the backprojection algorithm. For a demonstration of the omega-k algorithm, OpenCV will be required. You will most likely also want to use a UHF setup. To do this, simply change SARplatform to SARplatformUHF in the demo and comment/un-comment the bottom lines of the file as necessary. The platform dictionary keys defined in SARplatform and SARplatformUHF were taken from Carrera [1].

Also included in the ./examples folder are examples of how to process actual data. Full advantage is taken of Python's object oriented nature by including all system parameters in a platform object. As explained in the phsRead section, only a few steps are required to use the algorithms in this package for any arbitrary data set. They are:

1. read in the phase history and export as a NumPy array
2. read in the auxillary data
3. place auxillary data in a platform dictionary and export

The help files below detail what parameters need to be defined in the platform dictionary for a given function. For an arbitrary data source, the most rigorous task will just be renaming/deriving parameters already in the auxillary data and placing them in a platform dictionary. The source code for phsRead can be used as a reference. Included in the source code are examples of how to translate auxillary data from a .xml file (required for DIRSIG), a .mat file (required for the AFRL data set), and a binary file (required for the Sandia data set) into a python dictionary readable by RITSAR. Once the platform dictionary is defined, no additional work is needed use the algorithms in this package. Simply specify an image plane using imgTools.img_plane_dict() 3.4 or manually using the guidelines in 3.1.

2 phsTools

2.1 simulate_phs(platform, points, amplitudes)

This function is a basic phase history simulator. It takes a list of target locations and amplitudes and saves the demodulated signal to './phase_history.npy'. It also outputs the signal to the function caller. The amplitude of the received signal for a perfect point reflector is assumed to be one arbitrary unit. A uniform beam pattern is assumed and $1/r^4$ attenuation is not modeled.

Parameters:

- **platform** : dictionary
  dictionary that has the following keys defined:
  - **nsamples** : int
    number of fast time samples
  - **npulses** : int
    number of pulses
  - **chirprate** : float
    chirprate in Hz/s
  - **f_0** : float
    carrier frequency in Hz
  - **t** : float array [nsamples]
    demodulated fast-time centered at t=0. The array should go from $(-nsamples/2, nsamples/2) \times \Delta t$ where, $\Delta t = (\text{A/D sampling rate})^{-1}$
  - **pos** : float array [npulses x 3]
    (x,y,z) coordinates of platform, referenced to scene center, which is located at (0,0,0)
- **points** : float array [npoints x 3]
  locations of each point referenced to scene center
- **amplitudes** : float array [npoints]
  reflectance of each point. May be a complex number.

Returns:

- **phase_history** : complex array [npulses x nsamples]
  resultant phase history

2.2 RVP_correct(phs, platform)

Corrects Residual Video Phase using the formulation in Carrera Appendix C.

Parameters:

- **phs** : complex array [npulses x nsamples]
  complex phase history to be corrected for residual video phase
- **platform** : dictionary
  dictionary that has the following keys defined:
  - **nsamples** : int
    number of fast time samples
  - **npulses** : int
    number of pulses
  - **chirprate** : float
    chirprate in Hz/s
  - **delta_r** : float
    spacing between fast time samples. Defined here as $\frac{c}{2B_{IF}}$ where, B_{IF} is the bandwidth of the intermediate frequency (de-chirped) signal. B_{IF} is simply $(t_{max} - t_{min}) \times chirprate$ where t is the demodulated fast time, as defined in section 2.1.

Returns:

- **phase_history** : complex array [npulses x nsamples]
  resultant phase history, corrected for residual video phase

2.3 phs_to_const_ref(phs, platform, upchirp)

This program converts a phase history that was demodulated using a pulse dependant range to scene center to a phase history that is demodulated using a fixed reference. The fixed reference is defined as the minimum range to scene center.

Parameters:

- **phs** : complex array [npulses x nsamples]
  complex phase history that was compensated to a point
- **platform** : dictionary
  dictionary that has the following keys defined:
  - **npulses** : int
    number of pulses
  - **chirprate** : float
    chirprate in Hz/s
  - **f_0** : float
    carrier frequency in Hz
  - **t** : float array [nsamples]
    demodulated fast-time centered at t=0. The array should go from $(-nsamples/2, nsamples/2) \times \Delta t$ where, $\Delta t = (\text{A/D sampling rate})^{-1}$
  - **pos** : float array [npulses x 3]
    (x,y,z) coordinates of platform, referenced to scene center, which is located at (0,0,0)

Returns:

- **phase_history** : complex array [npulses x nsamples]
  resultant phase history, compensated to a line

3 imgTools

In this section the img_plane dictionary is introduced. A sample img_plane dictionary generator is provided in sarit/examples/dictionaries.

3.1 polar_format(phs, platform, img_plane, taylor)

This is the Polar Format algorithm. The phase history data as well as platform and image plane dictionaries are taken as inputs.

The phase history data is collected on a two-dimensional surface in k-space. For each pulse, a strip of this surface is collected. The first step in this program is to project each strip onto the (ku,kv) plane defined by the normal vector contained in the image plane dictionary. This will result in data that is unevenly spaced in (ku,kv). This unevenly spaced data is interpolated onto an evenly spaced (ku,kv) grid defined in the image plane dictionary. The interpolation is done along the radial direction first, then along the along-track direction. Further details of this method are given in both the Jakowitz and Carrera texts.

Parameters:

- **phs** : complex array [npulses x nsamples]
  complex phase history
- **platform** : dictionary
  dictionary that has the following keys defined:
  - **npulses** : int
    number of pulses
  - **f_0** : float
    carrier frequency in Hz
  - **pos** : float array [npulses x 3]
    (x,y,z) coordinates of platform, referenced to scene center, which is located at (0,0,0)
  - **k_r** : float array [nsamples]
    spatial frequencies corresponding to each fast time return. For a linear FM signal, this is given by $\frac{4\pi}{c}(f_0 + chirprate \times t)$
  - **R_c** : float
    vector to scene center at aperture center
- **img_plane** : dictionary
  dictionary that has the following keys defined:
  - **n_hat** : float array [3]
    unit normal vector to image plane
  - **k_u** : float array [size(u)]
    image plane spatial frequencies along u axis. u is defined to lie along the slant range vector, projected onto the image plane.
  - **k_v** : float array [size(v)]
    image plane spatial frequencies along v axis. v is defined to lie perpendicular to the slant range vector in the image plane.
- **taylor** : int
  attenuation factor for taylor window in dB. defaults to 43 dB.

Returns:

- **img** : complex array [npulses x nsamples]
  processed image

3.2 backprojection(phs, platform, img_plane, taylor, upsample)

This is the Backprojection algorithm. The phase history data as well as platform and image plane dictionaries are taken as inputs. The (x,y,z) locations of each pixel are required, as well as the size of the final image (interpreted as [size(v) x size(u)]).

Parameters:

- **phs** : complex array [npulses x nsamples]
  complex phase history
- **platform** : dictionary
  dictionary that has the following keys defined:
  - **nsamples** : int
    number of fast time samples
  - **npulses** : int
    number of pulses
  - **pos** : float array [npulses x 3]
    (x,y,z) coordinates of platform, referenced to scene center, which is located at (0,0,0)
  - **k_r** : float array [nsamples]
    spatial frequencies corresponding to each fast time return. For a linear FM signal, this is given by $\frac{4\pi}{c}(f_0 + chirprate \times t)$
  - **delta_r** : float
    spacing between fast time samples. Defined here as $\frac{c}{2B_{IF}}$ where, B_{IF} is the bandwidth of the intermediate frequency (de-chirped) signal. B_{IF} is simply $(t_{max} - t_{min}) \times chirprate$ where t is the demodulated fast time, as defined in section 2.1.
- **img_plane** : dictionary
  dictionary that has the following keys defined:
  - **u** : float array [len(u)]
    for the backprojection algorithm, u is essentially a placeholder that defines one of the dimmensions of the output image. Only the length of u matters, not its contents.
  - **v** : float array [len(v)]
    for the backprojection algorithm, v is essentially a placeholder that defines one of the dimmensions of the output image. Only the length of v matters, not its contents.
  - **pixel_locs** : float array [3 x n_pixels]
    flattened array specifying the 3-dimensional location of each pixel in the rectangular output array whose shape is [len(u) x len(v)]. Consequently, $n\_pixels = len(u) \times len(v)$. The locations do not have to lie in a plane and can be specified to an arbitrary surface that matches the local terrain. A rectangular output array with the location of each pixel specified is the only constraint.
- **taylor** : int
  attenuation factor for taylor window in dB. defaults to 43 dB.
- **upsample** : int
  upsampling factor, default is 6.

Returns:

- **img** : complex array [npulses x nsamples]
  processed image

3.3 omega-k(phs, platform, taylor, upsample)

This is an omega-k algorithm based off of the algorithm prescribed in the Carrera text. Only the phase history and platform files are taken as inputs, an img_plane dictionary is not required.

The input phase history needs to have been demodulated to a fixed reference. If demodulated to scene center, the included phs_const_ref file can do the conversion for you. A straight line flight path is also assumed.

The first step in the algorithm is to perform a 1D FT along azimuth. A matched filter is applied to the resultant data to perfectly compensate the range curvature of all scatterers having minimum range R_s. The default setting for R_s is the minimum range of scene center. To correct the range curvature for other scatterers, the data is mapped onto a new grid.

Parameters:

- **phs** : complex array [npulses x nsamples]
  complex phase history
- **platform** : dictionary
  dictionary that has the following keys defined:
  - **nsamples** : int
    number of fast time samples
  - **npulses** : int
    number of pulses
  - **pos** : float array [npulses x 3]
    (x,y,z) coordinates of platform, referenced to scene center, which is located at (0,0,0)
  - **k_r** : float array [nsamples]
    spatial frequencies of demodulated signal corresponding to each fast time return. For a linear FM signal, this is given by $\frac{4\pi}{c}(f_0 + chirprate \times t)$
  - **k_y** : float
    spatial frequencies along flight path. A straight line flight path (or phase history motion compensated to a straight line flight path) are required.
- **taylor** : int
  attenuation factor for taylor window in dB. defaults to 43 dB.
- **upsample** : int
  upsampling factor, default is 6.

Returns:

- **img** : complex array [npulses x nsamples]
  processed image

3.4 img_plane_dict(platform, res_factor, n_hat, aspect, upsample)

This function defines the image plane parameters. The user specifies the image resolution using the res_factor. A res_factor of 1 yields a (u,v) image plane whose pixels are sized at the theoretical resolution limit of the system (derived using delta_r which in turn was derived using the bandwidth). The user can also adjust the aspect of the image grid. This defaults to nsamples/npulses.

'n_hat' is a user specified value that defines the image plane orientation w.r.t. to the nominal ground plane.

Parameters:

- **platform** : dictionary
  dictionary that has the following keys defined:
  - **nsamples** : int
    number of fast time samples
  - **npulses** : int
    number of pulses
  - **R_c** : float array [3]
    vector to scene center at aperture center
  - **delta_r** : float array [nsamples]
    spacing between fast time samples. Defined here as $\frac{c}{2B_{IF}}$ where, B_{IF} is the bandwidth of the intermediate frequency (de-chirped) signal. B_{IF} is simply $(t_{max} - t_{min}) \times chirprate$ where t is the demodulated fast time, as defined in section 2.1.
- **res_factor** : float
  image resolution in units of theoretical resolution size. This defaults to 1.0.
- **n_hat** : float array [3]
  unit normal vector to image plane. Defaults to (0,0,1).
- **aspect** : float
  aspect ratio of range to cross range. Determines cross-range resolution by multiplying res_factor by aspect. Defaults to nsamples/npulses
- **upsamples** : bool
  Option to upsample number of pixels. The number of pixels in the range direction is set to a power of 2 that is one power of 2 higher than nsamples. If nsamples is already a power of 2, the number of range pixels is just set to nsamples. The same goes for npulses and number of cross-range pixels.

Returns:

- **img_plane** : dictionary
  dictionary that has the following keys defined:
  - **n_hat** : float array [3]
    unit normal vector to image plane. Defaults to (0,0,1).
  - **u** : float array [len(u)]
    pixel locations in range direction. u is defined to lie along the slant range vector, projected onto the image plane.
  - **v** : float array [len(v)]
    pixel locations in cross-range direction. v is defined to lie perpendicular to the slant range vector in the image plane.
  - **du** : float array [len(u)]
    pixel spacing in range direction
  - **dv** : float array [len(v)]
    pixel spacing in cross-range direction

- **k_u** : float array [size(u)]
  image plane spatial frequencies along u axis. u is defined to lie along the slant range vector, projected onto the image plane.
- **k_v** : float array [size(v)]
  image plane spatial frequencies along v axis. v is defined to lie perpendicular to the slant range vector in the image plane.
- **pixel_locs** : float array [3 x n_pixels]
  flattened array specifying the 3-dimensional location of each pixel in a rectangular output array whose shape is [len(u) x len(v)]. Consequently, $n\_pixels = len(u) \times len(v)$. The locations do not have to lie in a plane and can be specified to an arbitrary surface that matches the local terrain. A rectangular output array with the location of each pixel specified is the only constraint. If the image plane is specified in this manner, only a time domain algorithm (backprojection) can be used to process the data.

3.5 autoFocus(img, win, win_params)

This program autofocuses an image using the Phase Gradient Algorithm. If the parameter win is set to auto, an adaptive window is used. Otherwise, the user sets win to 0 and defines win_params. The first element of win_params is the starting windows size. The second element is the factor by which to reduce it by for each iteration. This algorithm is based off [2].

Parameters:

- **img** : complex array [npulses x nsamples]
  complex image
- **win** : string
  if 'auto', an adaptive window is used. Otherwise, win_params is used to create the window. Default is 'auto.'
- **win_params** : list
  The first element of win_params is the starting windows size. The second element is the factor by which to reduce it by for each iteration.

Returns:

- **img_af** : complex array [npulses x nsamples]
  autofocused image

4 phsRead

This section describes the RITSAR interfaces to various datasets. In a nutshell, the user supplies the directory containing the phase history and auxillary data. The functions below essentially perform 3 tasks:

1. read in the phase history and export as a NumPy array
2. read in the auxillary data
3. place auxillary data in a platform dictionary and export

As shown in the functions below, code can be written to handle many different data formats including .envi, .mat, .xml, and binary. As of now, RITSAR can interface with DIRSIG and AFRL Gotcha data. A function is also provided that processes a Sandia data set. These functions serve mostly to show that data in almost any format can be processed with the RITSAR toolset provided code is written that performs the three tasks listed above for a given format.

4.1 DIRSIG(directory)

This function reads in the DIRSIG xml data as well as the envi header file from the user supplied directory. The phs and a Python dictionary compatible with RITSAR are returned to the function caller. More information on DIRSIG's SAR capability can be found here: http://www.dirsig.org/docs/new/radar.html

Parameters:

- **directory** : string
  string containing directory of DIRSIG data to include the xml and envi header files.

Returns:

- **phase_history** : complex array [npulses x nsamples]
  the complex phase history
- **platform** : dictionary
  dictionary that has the following keys defined:
  - **f_0** : float
    carrier frequency in Hz
  - **chirprate** : float
    chirprate in Hz/s
  - **nsamples** : int
    number of fast time samples
  - **npulses** : int
    number of pulses
  - **B** : float
    bandwidth of transmitted signal
  - **B_IF** : float
    bandwidth of intermediate frequency, demodulated signal.
  - **delta_r** : float
    spacing between fast time samples. Defined here as $\frac{c}{2B_{IF}}$ where, B_{IF} is the bandwidth of the intermediate frequency (de-chirped) signal. B_{IF} is simply $(t_{max} - t_{min}) \times chirprate$ where t is the demodulated fast time, as defined in section 2.1.
  - **delta_t** : float
    time between fast time samples. Defined as (A/D sampling rate)$^{-1}$
  - **vp** : float
    velocity of platform in m/s

- **pos** : float array [npulses x 3]
  (x,y,z) coordinates of platform, referenced to scene center, which is located at (0,0,0)
- **R_c** : float array [3]
  vector to scene center at aperture center
- **t** : float array [nsamples]
  demodulated fast-time centered at t=0. The array should go from $(-nsamples/2, nsamples/2) \times \Delta t$ where, $\Delta t = (\text{A/D sampling rate})^{-1}$
- **k_r** : float array [nsamples]
  spatial frequencies of demodulated signal corresponding to each fast time return. For a linear FM signal, this is given by $\frac{4\pi}{c}(f_0 + chirprate \times t)$
- **k_y** : float
  spatial frequencies along flight path. A straight line flight path (or phase history motion compensated to a straight line flight path) are required.
- **metadata** : string array
  xml metadata

5 AFRL(directory, pol, start_az, n_az=3)

This function reads in the AFRL Gotcha *.mat files from the user supplied directory and exports both the phs and a Python dictionary compatible with RITSAR.
Parameters:

- **directory** : string
  string containing directory of DIRSIG data to include the xml and envi header files.
- **pol** : string
  polarization to be processed. For example, 'HH'.
- **start_az** : int
  starting azimuth (0 to 360)
- **n_az** : int
  number of azimuth files to process. Default is 3.

Returns:

- **phase_history** : complex array [npulses x nsamples]
  the complex phase history
- **platform** : dictionary
  dictionary that has the following keys defined:
  - **f_0** : float
    carrier frequency in Hz
  - **chirprate** : float
    chirprate in Hz/s
  - **nsamples** : int
    number of fast time samples
  - **npulses** : int
    number of pulses
  - **B_IF** : float
    bandwidth of intermediate frequency, demodulated signal.
  - **pos** : float array [npulses x 3]
    (x,y,z) coordinates of platform, referenced to scene center, which is located at (0,0,0)

- **delta_r** : float
  spacing between fast time samples. Defined here as $\frac{c}{2B_{IF}}$ where, B_{IF} is the bandwidth of the intermediate frequency (de-chirped) signal. B_{IF} is simply $(t_{max} - t_{min}) \times chirprate$ where t is the demodulated fast time, as defined in section 2.1.
- **R_c** : float array [3]
  vector to scene center at aperture center
- **t** : float array [nsamples]
  demodulated fast-time centered at t=0. The array should go from $(-nsamples/2, nsamples/2) \times \Delta t$ where, $\Delta t = (\text{A/D sampling rate})^{-1}$
- **af_r** : float array [npulses]
  column vector containing the correction for R_c
- **af_ph** : float array [npulses]
  column vector containing the phase correction
- **k_r** : float array [nsamples]
  spatial frequencies of demodulated signal corresponding to each fast time return. For a linear FM signal, this is given by $\frac{4\pi}{c}(f_0 + chirprate \times t)$
- **(k_y)** : float
  not currently available (due to circular flight path). may be added later when moComp_to_line is complete.

5.1 Sandia(directory)

This function reads in the Sandia *.phs and *.au2 files from the user supplied directoryand exports both the phs and a Python dictionary compatible with RITSAR.
Parameters:

- **directory** : string
  string containing directory of DIRSIG data to include the

Returns:

- **phase_history** : complex array [npulses x nsamples]
  the complex phase history
- **platform** : dictionary
  dictionary that has the following keys defined:
  - **f_0** : float
    carrier frequency in Hz
  - **chirprate** : float
    chirprate in Hz/s
  - **nsamples** : int
    number of fast time samples
  - **npulses** : int
    number of pulses
  - **B_IF** : float
    bandwidth of intermediate frequency, demodulated signal.
  - **pos** : float array [npulses x 3]
    (x,y,z) coordinates of platform, referenced to scene center, which is located at (0,0,0)
  - **delta_r** : float
    spacing between fast time samples. Defined here as $\frac{c}{2B_{IF}}$ where, B_{IF} is the bandwidth of the intermediate frequency (de-chirped) signal. B_{IF} is simply $(t_{max} - t_{min}) \times chirprate$ where t is the demodulated fast time, as defined in section 2.1.

- **R_c** : float array [3]
  vector to scene center at aperture center
- **t** : float array [nsamples]
  demodulated fast-time centered at t=0. The array should go from $(-nsamples/2, nsamples/2) \times \Delta t$ where, $\Delta t = (\text{A/D sampling rate})^{-1}$
- **k_r** : float array [nsamples]
  spatial frequencies of demodulated signal corresponding to each fast time return. For a linear FM signal, this is given by $\frac{4\pi}{c}(f_0 + chirprate \times t)$ added later when moComp_to_line is complete.

References

[1] W. G. Carrera, R. S. Goodman, and R. M. Majewski, *Spotlight Synthetic Aperture Radar*. Artech House, Inc., 1995.

[2] D. Wahl, P. Eichel, D. Ghiglia, and J. Jakowatz, C.V., "Phase gradient autofocus-a robust tool for high resolution sar phase correction," *Aerospace and Electronic Systems, IEEE Transactions on*, vol. 30, pp. 827–835, Jul 1994.