pyroSAR - A Framework for Large-Scale SAR Satellite Data Processing

Poster · May 2019

DOI: 10.13140/RG.2.2.16424.83206

4 authors:

John Truckenbrodt
Friedrich Schiller University Jena
9 PUBLICATIONS **49** CITATIONS
SEE PROFILE

Felix Cremer
Friedrich Schiller University Jena
9 PUBLICATIONS **23** CITATIONS
SEE PROFILE

Ismail Baris
German Aerospace Center (DLR)
13 PUBLICATIONS **6** CITATIONS
SEE PROFILE

Jonas Eberle
Friedrich Schiller University Jena
27 PUBLICATIONS **286** CITATIONS
SEE PROFILE

PYROSAR – A FRAMEWORK FOR LARGE-SCALE SAR SATELLITE DATA PROCESSING

John Truckenbrodt[1,2], Felix Cremer[1,2], Ismail Baris[3], Jonas Eberle[1]*

[1]Friedrich-Schiller-University Jena, Institute of Geography, Department for Earth Observation, Jena, Germany

[2]German Aerospace Centre DLR, Institute of Data Science, Jena, Germany

[3]German Aerospace Centre DLR, Microwaves and Radar Institute, Oberpfaffenhofen, Germany

*john.truckenbrodt@uni-jena.de

Why PYROSAR?

pyroSAR is intended for making SAR data processing and formatting as easy as possible. With pyroSAR, managing all available data in a central location becomes a breeze. You want to know what data from various SAR satellite missions is available for your test site on your group's server and process all scenes to the extent of that site? pyroSAR has got you covered. It reads a large number of SAR image formats, extracts metadata and stores it in a database, offers APIs to process data with either ESA SNAP or GAMMA, and keeps track of consistently processed data. Furthermore, several tools are available to handle the processed datasets and conveniently export them to other software.

Metadata Handling

SAR scene attributes like the acquisition mode, orbit and spatial extent can be read from various SAR image formats like Sentinel-1 SAFE, CEOS and TerraSAR-X. The different image formats are scanned for relevant content and translated to homogenized metadata field names, e.g. the entry *MPH_SENSING_START* in ESA format to the format-independent name *start*. This way each scene's metadata can be addressed in the same way and all attributes can easily be inserted into a database. The database functionality currently uses sqlite + spatialite as a lightweight option for writing a single database file.

```
>>> from pyroSAR import identify
>>> filename = 'S1A_IW_GRDH_1SDV_20180829T170656_20180829T170721_023464_028DE0_F7BD.zip'
>>> scene = identify(filename)
>>> print(scene)
pyroSAR ID object of type SAFE
acquisition_mode: IW
cycleNumber: 148
frameNumber: 167392
lines: 16703
orbit: A
orbitNumber_abs: 23464
orbitNumber_rel: 117
polarizations: ['VV', 'VH']
product: GRD
projection: +proj=longlat +datum=WGS84 +no_defs
samples: 26056
sensor: S1A
spacing: (10.0, 10.0)
start: 20180829T170656
stop: 20180829T170721
```

```
>>> from pyroSAR import Archive, identify
>>> from spatialist.ancillary import finder
>>> dbfile = '/.../scenelist.db'
>>> archive_s1 = '/.../sentinel1/GRD'
>>> scenes_s1 = finder(archive_s1, ['^S1[AB].*\.zip'], regex=True, recursive=True)
>>> with Archive(dbfile) as archive:
>>>     archive.insert(scenes_s1)
```

```
>>> from pyroSAR import Archive
>>> from spatialist import Vector
>>> archive = Archive('/path/to/dbfile.db')
>>> site = Vector('/path/to/site.shp')
>>> outdir = '/path/to/processed/results'
>>> maxdate = '20171231T235959'
>>> selection_proc = archive.select(vectorobject=site, processdir=outdir,
>>>                                 maxdate=maxdate, sensor=('S1A', 'S1B'),
>>>                                 product='GRD', acquisition_mode='IW', vv=1)
>>> archive.close()
```

SNAP API

pyroSAR's SNAP functions directly parse XML processing workflows and execute them over SNAP's Graph Processing Tool (GPT). The workflow is flexibly adapted depending on the user input. For example, a vector file can be passed to use its bounding box for processing only a subset of the SAR scene. The vector file is reprojected in-memory if necessary. The API passes further command line arguments to the GPT call and reformats the resulting images to ensure a consistent output for convenient further processing. The custom SNAP workflow is stored together with the output products to keep track of how a scene was processed. The names are automatically parsed from the input product and the performed processing steps.

```
>>> from pyroSAR.snap import geocode
>>> filename = 'S1A_IW_GRDH_1SDV_20180829T170656_20180829T170721_023464_028DE0_F7BD.zip'
>>> geocode(infile=filename, outdir='outdir', tr=20, scaling='dB',
>>>         export_extra=['DEM', 'localIncidenceAngle'], t_srs=4326)
```

```
S1A__IW___A_20180829T170656_bnr_Orb_Cal_TF_TC_dB_proc.xml
S1A__IW___A_20180829T170656_dem.tif
S1A__IW___A_20180829T170656_localIncidenceAngle.tif
S1A__IW___A_20180829T170656_VH_bnr_Orb_Cal_TF_TC_dB.tif
S1A__IW___A_20180829T170656_VV_bnr_Orb_Cal_TF_TC_dB.tif
```

GAMMA API

pyroSAR offers a Python API to conveniently process SAR scenes without worrying about the numerous Gamma command line tools and their parametrization. All tools are wrapped into Python functions using pyroSAR's own documentation parser. All parsed functions take a directory for writing log files and a shell script name for documenting the executed commands as input. Several convenience functions exist that execute a series of Gamma commands via a convenient Python interface, like *dem_autocreate* to automatically download DEM files and convert them to Gamma format or *geocode* for processing scenes to RTC backscatter.

Gamma command call in Linux bash

```
offset_fit offs ccp off.par coffs - 0.15 3 0 > offset_fit.log
```

vs.

executing the parsed command in Python

```
import os
from pyroSAR.gamma.api import isp

workdir = '/data/gamma_workdir'

parameters = {'offs': os.path.join(workdir, 'offs'),
              'ccp': os.path.join(workdir, 'ccp'),
              'OFF_par': os.path.join(workdir, 'off.par'),
              'coffs': os.path.join(workdir, 'coffs'),
              'thres': 0.15,
              'npoly': 3,
              'interact_flag': 0,
              'logpath': workdir}

isp.offset_fit(**parameters)
```

pyroSAR.gamma.parser_demo.**offset_fit**(*offs, ccp, OFF_par, coffs='-', coffsets='-', thres='-', npoly='-', interact_mode='-', logpath=None, outdir=None, shellscript=None*) [source]

Range and azimuth offset polynomial estimation
Copyright 2011, Gamma Remote Sensing, v3.3 28-Nov-2015 clw/uw

readthedocs example

```
>>> from pyroSAR.gamma import geocode
>>> filename = 'S1A_IW_GRDH_1SDV_20180829T170656_20180829T170721_023464_028DE0_F7BD.zip'
>>> geocode(scene=filename, dem='demfile', outdir='outdir', targetres=20, scaling='db',
>>>         export_extra=['dem_seg_geo', 'inc_geo', 'ls_map_geo'])
```

```
S1A__IW___A_20180829T170656_commands.sh
S1A__IW___A_20180829T170656_dem_seg_geo.tif
S1A__IW___A_20180829T170656_inc_geo.tif
S1A__IW___A_20180829T170656_ls_map_geo.tif
S1A__IW___A_20180829T170656_manifest.safe
S1A__IW___A_20180829T170656_VH_grd_mli_norm_geo_db.tif
S1A__IW___A_20180829T170656_VV_grd_mli_norm_geo_db.tif
```

Spatial Data Tools

A collection of tools is available for downloading and handling ancillary data needed for processing, DEMs and orbit state vector files in particular, as well as further modifying the processing result. This way all data can be prepared before processing jobs are dispatched to server nodes without internet access. General spatial data handling and processing is realized via a GDAL API packages spatialist, which is currently developed together with pyroSAR.

mosaic and stack processed results to a single file

```
from pyroSAR.ancillary import groupbyTime, find_datasets, seconds
from spatialist.raster import stack

# find pyroSAR files by metadata attributes
archive_s1 = '/.../sentinel1/GRD/processed'
scenes_s1 = find_datasets(archive_s1, sensor=('S1A', 'S1B'), acquisition_mode='IW')

# group images by acquisition time
groups = groupbyTime(images=scenes_s1, function=seconds, time=30)

# mosaic individual groups and stack the mosaics to a single ENVI file
# only files overlapping with the shapefile are selected and resampled to its extent
stack(srcfiles=groups, dstfile='stack', resampling='bilinear', targetres=(20, 20),
      srcnodata=-99, dstnodata=-99, shapefile='site.shp', separate=False)
```

Prepare a SRTM mosaic for a Sentinel-1 scene

```
from pyroSAR import identify
from pyroSAR.auxdata import dem_autoload
from spatialist import gdalwarp

# identify the SAR scene
filename = 'S1A_IW_SLC__1SDV_20150330T170734_20150330T170801_005264_006A6C_DA69.zip'
scene = identify(filename)

# extract the bounding box as spatialist.Vector object
bbox = scene.bbox()

# download the tiles and virtually combine them in an in-memory
# VRT file subsetted to the extent of the SAR scene plus a buffer of 0.01 degrees
vrt = dem_autoload(geometries=[bbox], demType='SRTM 1Sec HGT',
                   vrt='/vsimem/srtm1.vrt', buffer=0.01)

# write the final GeoTiff file
outname = scene.outname_base() + 'srtm1.tif'
gdalwarp(src=vrt, dst=outname, options={'format': 'GTiff'})
```

Data Cube Export

pyroSAR offers tools for creating YAML files, which can be passed to the Open Data Cube command line tools to ingest processed SAR data into a cube. A group of processed SAR files can be generalized to an ODC product, which can then be created via an exported YAML file. Any new product which is to be added to the data cube product is automatically checked against the product specifications.

```
from pyroSAR.datacube_util import Product, Dataset
from pyroSAR.ancillary import find_datasets

# find pyroSAR files by metadata attributes
archive_s1 = '/.../sentinel1/GRD/processed'
scenes_s1 = find_datasets(archive_s1, sensor=('S1A', 'S1B'), acquisition_mode='IW')

# define the polarization units describing the data sets
units = {'VV': 'backscatter VV', 'VH': 'backscatter VH'}

# create a new product
with Product(name='S1_GRD_index',
             product_type='gamma0',
             description='Gamma Naught RTC backscatter') as prod:

    for dataset in scenes_s1:
        with Dataset(dataset, units=units) as ds:

            # add the dataset to the product
            prod.add(ds)

            # parse datacube indexing YMLs from product and data set metadata
            prod.export_indexing_yml(ds, 'yml_index_outdir')

    # write the product YML
    prod.write('yml_product')

    # print the product metadata which is written to the product YML
    print(prod)
```

Software Maintenance

The code is maintained on GitHub under a MIT license thus allowing use without restrictions and liability as long as credit is given. The current version hosted on the Python Package Index (PyPI) is 0.8. To install it, use

python -m pip install pyroSAR

A documentation framework is maintained using Sphinx, which is hosted on https://pyrosar.readthedocs.io. All code examples can also be found online.

ID	Abstract class for SAR meta data handlers
CEOS_PSR	Handler class for ALOS-PALSAR data in CEOS format
CEOS_ERS	Handler class for ERS data in CEOS format
ESA	Handler class for SAR data in ESA format (Envisat ASAR, ERS-1/2)
SAFE	Handler class for Sentinel-1 data
TSX	Handler class for TerraSAR-X and TanDEM-X data

pyroSAR accommodates a testing framework using pytest, which ensures that the code is working as expected. Once new commits are pushed to GitHub, all tests are executed on the continuous integration services AppVeyor and Travis CI for ensuring functionality on Windows and Linux respectively. The code test coverage is reported to coveralls for keeping track of which code is not yet covered by the tests.
If you're interested in the project, find issues, have further feature requests or want to contribute head on over to the project's web page on GitHub:

https://github.com/johntruckenbrodt/pyroSAR

Outlook

Several features are still in the making or are considered for the future:

- Export to the Earth System Data Lab (ESDL) Data Cube
- Export to GRASS GIS
- Processing with Orfeo Toolbox
- Workflows for SLC data handling and SAR Interferometry
- …

FRIEDRICH-SCHILLER-UNIVERSITÄT JENA