User Manual

DInSAR geocoding library and
execution script

TABLE OF CONTENTS

1. PREFACE

This manual presents the python library and execution script developed with the Earth Observation Lab in the University of the Bundeswehr in Munich. Its aim is to better understand the code provided and present an easy utilisation for users.

2. DESCRIPTION OF THE PRODUCT

F*igure 1* describes the processing pipeline used in the execution script. It has been deduced from the base processing chain proposed in the SNAP prebuilt graphs.

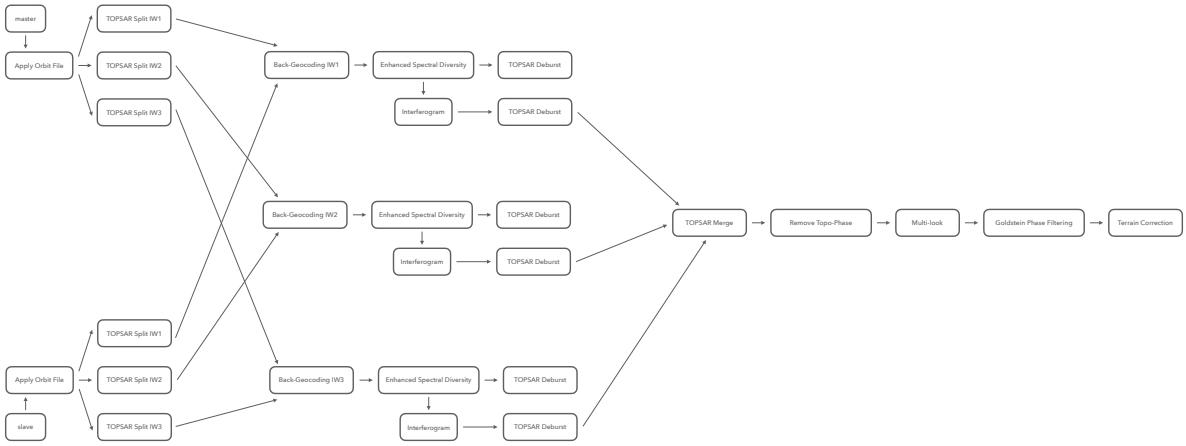

Fig. 1

The master image and every slave image are split after having applied the orbit files. For each split, backgeocoding, enhanced spectral diversity as well as deburst functions are applied. Meanwhile are applied interferogram generation and deburst functions after the backgeocoding. The interferograms are then merged and topographic phase removal, multilook, phase filtering and geocoding functions are applied.

2.1 DInSAR geocoding library

F*igure 2* describes the overall file structure of the project. The executing script is found in the singular *main.py* file whilst the dedicated python library can be found under the *dinsargeocoding* folder.

The choice to build a dedicated python library come from the need to simplify the use of the SnapPy library than can prove tedious at times. Functions can be preparametered in the library then directly used inside various scripts.

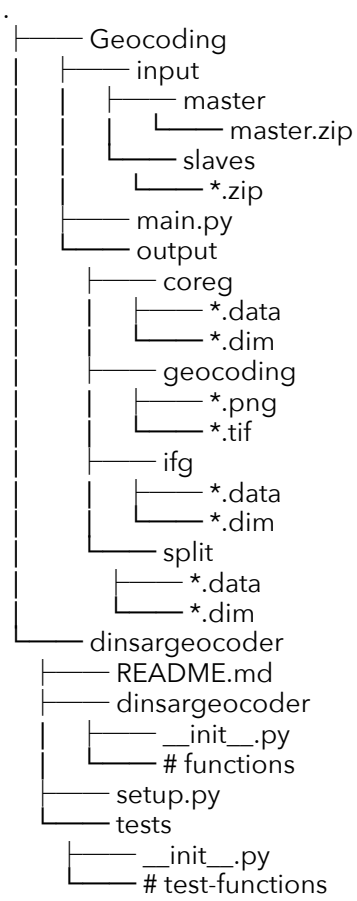

Fig. 2

2.2 Environment

To run the different functions, it is necessary to have a machine capable of running Python 3.6 < 3.8 as other version are incompatible with SnapPy.

Most of SnapPy processes are quite heavy and can present to be quite time consuming. To avoid any unnecessary time loss it is recommended to have a powerful machine capable of handling heavy memory loads.

3. INSTALLATION

Most of the dependencies for this project are Python librairies. They can be installed easily using *pip* the most difficult part remaining is the SnapPy and GDAL libraries installation.

3.1 Create a Python environment

It is recommended to create Python environment of version either 3.6 or higher, or 3.8 or lower.

It is possible to use Anaconda or any other environment.

3.2 Install SNAP

If you are using a personal machine you can easily follow the instructions on the official ESA SNAP website:

https://senbox.atlassian.net/wiki/spaces/SNAP/pages/50855941/Configure+Python+to+use+the+SNAP-Python+snappy+interface

If you are using a remote server you can refer to the following official documentation and install the SNAP command line version:

https://senbox.atlassian.net/wiki/spaces/SNAP/pages/30539778/Install+SNAP+on+the+command+line

3.3 Install SnapPy and GDAL

In order to install the SnapPy library on the more recent versions of Python the *jpy* wheel must either be built manually or downloaded from another source. After copying the wheel into the correct SNAP install directory, it possible to reset the SnapPy paths using the *snappy-conf* file. For an easier use it is also possible to set an environment variable to point to the correct folder.

Directly using *pip* to install GDAL posed to be more complicated than a standard install. It is recommended to first manually build or download the GDAL wheel and then install the library using the newly acquired wheel.

3.4 Other dependencies

If any other of the required dependencies cannot be found on your system a prompt will appear and you can simply install them using *pip* without any issues.

4. USAGE

The project is divided into two distinct parts, the dedicated library found under the *dinsargeocoder* folder and the *main.py* execution file.

4.1 Build and install the library

The *dinsargeocoder* library can be built and installed like a typical library using the following commands:

```
python3.8 setup.py bdist_wheel
python3.8 -m pip install . –user
```

4.2 Execution script

Once the dedicated library has been install the *main.py* execution script can be launched using the following command:

```
python3.8 main.py
```

Note that it is required that the starting file structure looks similar to the *Figure 3* before launching the script:

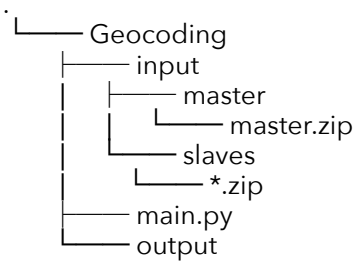

Fig. 3

The *master* folder need to contain the master image in a *.zip* file format and the *slaves* folder need to contain the slave images in a *.zip* file format.

5. MAINTENANCE

Since most dependencies come from Python libraries, they can be easily managed.

5.1 Dependencies and Librairies

The dependencies, if outdated, will print a warning and can then be updated via *pip*. As SnapPy is a fairly new library it is expected to receive several updates in the future, thus making a backup before updating any library is recommended for compatibility issues.

5.2 Evolution

It is possible to modify the parameters of the processing functions directly within the library, if so the library will have to be rebuilt before any new uses. Furthermore, it is possible to consider adding new functions for an easier use.

6. TROUBLESHOOTING

The most common issues that can arise while using the app are memory and slowness issues.

If processing two images takes up more than an hour, it can be recommended to increase the memory size. The most efficient way to fix this problem is to increase either or both the *java_max_mem* and the *tileCacheSize* parameters. While the *java_max_mem* setting will in most cases automatically set itself to around 70% of the total memory of the machine, the *tileCacheSize* setting however it automatically set to 1024Mo. This value can be modified to increase the performance drastically. However, whilst it can be set at around 70% of the *java_max_mem* setting value I would advise to to go past 16 384Mo as it would cause further memory overload problems.

Some warnings can occur during the processes such as *Java memory heap* which are caused by the memory not being cleared fast enough in-between the different function calls. Whilst it is not an optimal situation to wait for the memory to be cleared automatically, it is not an error that will cause the processes to fail.