Fusion Approach for Remotely-Sensed Mapping of Agriculture (FARMA): Tutorial (Lite)

Nathan Thomas*, Christopher R. S. Neigh, Mark L. Carroll, Jessica L. McCarty, Pete Bunting
* nmthomas28@gmail.com

FARMA is a framework for efficiently merging VHR-derived objects with commonly available remotely-Sensed imagery for time-series mapping and monitoring of agriculture at the field-scale across large geographical areas. FARMA allows the fusion of information from a wide range of image resolutions using distributed computing architecture where available. FARMA is built upon a number of Python packages including RSGISLib, Geopandas and Numpy.

Our publication on FARMA can be found here: https://www.mdpi.com/2072-4292/12/20/3459

This tutorial is designed as a quick set of instructions to guide you through the very basics of FARMA using the dataset example provided here: https://github.com/nmt28/FARMA/tree/main/Datasets. This tutorial assumes that the user has at least some knowledge of python and provides a basic overview of the scripts that need to be run to populate objects with time-series raster data. No analysis is demonstrated in this tutorial.

Software

FARMA is primarily based upon the Remote sensing and GIS Library, a python library that contains over 400 commands for vector and raster processing. RSGISLib can be found here: https://www.rsgislib.org. RSGISLib is available through Docker here: https://hub.docker.com/r/petebunting/au-eoed. Installation instructions and 'how to guide' can be found here: https://www.rsgislib.org/download.html and here: https://remotesensing.info/tutorialmaterials/au-eoed_tools_docker.pdf. RSGISLib is also available through Anaconda using the conda-forge channel.

Overview

This tutorial will demonstrate the steps required to implement FARMA. This includes the following steps:

- Clumping of an input segmentation
- Preparation of the segmentation for tiling
- Tiling of the segmentation and vectorization
- Population of the objects with raster values

FARMA does not generate segmentations although one is provided for you for the purpose of this tutorial.

Each of these scripts was built and run on a MacBook Pro (3.5 GHz Dual-Core Intel Core i7) with 16GB RAM

When installing RSGISLib, we also recommend using TuiView for viewing the datasets: http://tuiview.org. This can be installed via Anaconda (conda-forge) and launched from the terminal using the command "TuiView".

FARMA

Step 1: Clumping the Segmentation (0_ClumpSegmentation.py)

Code: https://github.com/nmt28/FARMA/blob/main/Code/0_ClumpSegmentation.py
Dataset:
https://github.com/nmt28/FARMA/blob/main/Datasets/WV03_20170530_M1BS_104001002EBA0900-toa_ND_k80_n4000_segs_tiles.kea

0_ClumpSegmentation.py is designed to clump an input image into objects (if required) and to generate the KEA format files as required for input into FARMA. RSGISLib generated segmentations do not require this step. External segmentations should be provided as images where each object is a homogenous group of pixels of the same value, such as in the example dataset provided.

0_CLumpSegmentation.py includes 4 different clumping approaches to choose from. The simplest is "CLUMP_RAM" which uses a single computing core and processes the segmentation in memory. This is the primary choice unless RAM or image size restrictions prevent this. A modified "CLUMP_DISK" can also be used which uses a single computing core but processes the segmentation on disk, where RAM is limited. For very large segmentations, "TILED_SINGLE" and "TILED_MULTI" are available. These tile the segmentation for clumping, suing either one or multiple cores. Additional options are required to run these two algorithms, as outlined below.

For its simplest use (and for use in this tutorial):

```
Python 0_ClumpSegmentation.py -i WV03_20170530_M1BS_104001002EBA0900-toa_ND_k80_n4000_segs_tiles.kea -m CLUMP_RAM
```

If the segmentation needs to be clumped on the Disk, use:

Python 0_ClumpSegmentation.py -i WV03_20170530_M1BS_104001002EBA0900-toa_ND_k80_n4000_segs_tiles.kea -m CLUMP_DISK

When using TILED_SINGLE a tile size must be provided in pixels (in one dimension):

Python 0_ClumpSegmentation.py -i WV03_20170530_M1BS_104001002EBA0900-toa_ND_k80_n4000_segs_tiles.kea -m TILED_SINGLE -t 1000

When using TILED_MULTI the number of computing cores must also be provided:

Python 0_ClumpSegmentation.py -i WV03_20170530_M1BS_104001002EBA0900-toa_ND_k80_n4000_segs_tiles.kea -m TILED_MULTI -t 1000 -c 4

This will generate a new image in the input directory and will have “clump.kea” appended to the file name. Using TuiView, the output image will look similar to the Figure 1 (note: TuiView is just for visualization of the KEA file and colors of objects are randomly assigned).

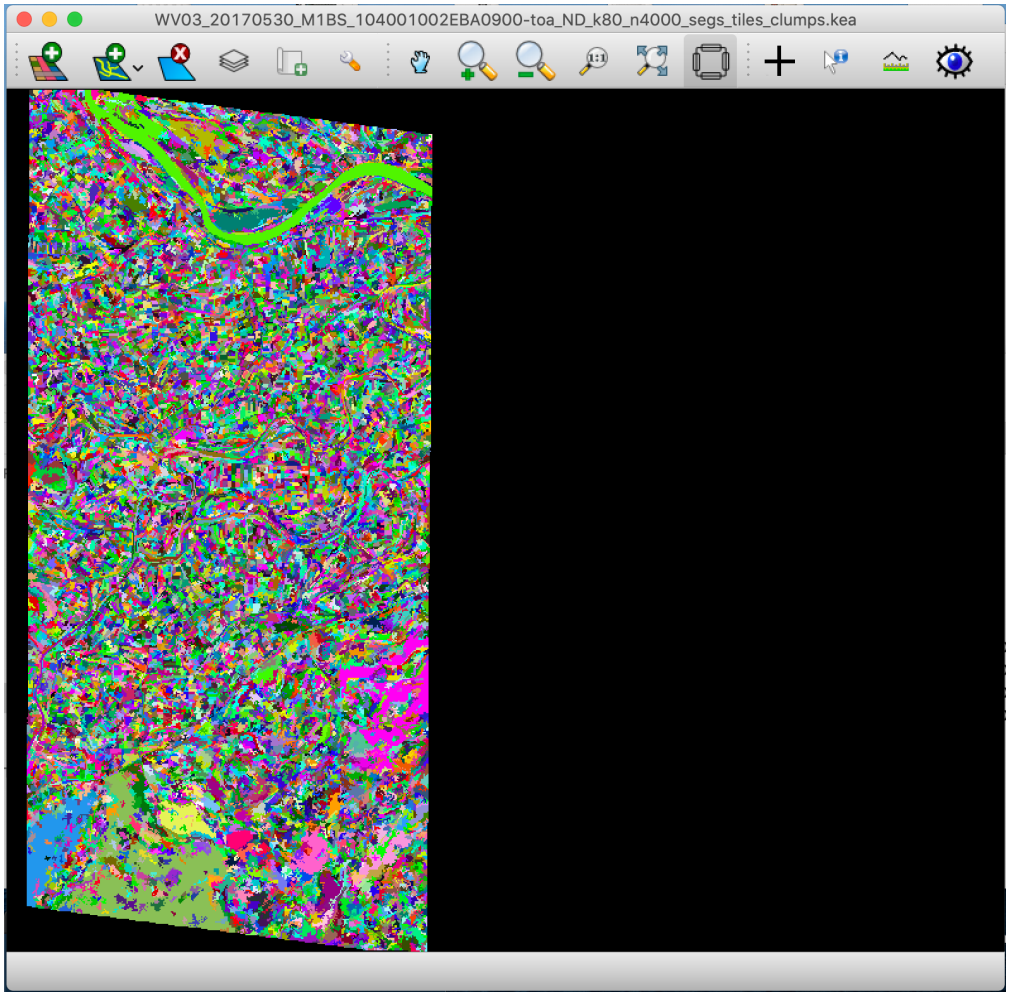

Figure 1. KEA format segmentation in TuiView

Step 2: Preparing the clumped segmentation for tiling (1_CreateRegGrid.py)
A key component of FARMA is an ability to tile a segmentation for efficient processing across large geographical domains. This step prepares the clumped segmentation from Step 1 for this process by creating a grid based on the extent of the input segmentation and a user defined tile size, before populating the mode of the grid cell into the segmentation as well as the spatial extent of each object. For processing in step 3, an image of the mode populated objects is also exported during this step. This script is run on a single thread and can be time intensive for VERY large segmentations. If the data used is prohibitively large, data should be tiled before use.

The script is run as follows, using the following command options:

-i input clumped segmentation from step 1
-t user defined tile size in pixels in one dimension (this will vary depending on the number of tiles required)

Python 1_CreateRegGrid.py -i WV03_20170530_M1BS_104001002EBA0900-toa_ND_k80_n4000_segs_tiles_clump.kea -t 1000

This will generate two new files:

1. A Regular Grid file (in the input directory appended with "_regGrid.kea"
2. The mode object image (in the input directory appended with "_modeTileMsk.kea"

Step 3: Segmentation Tiling and Vectorization (2_BoundingBoxes_Docker.py)

Step 3 involves the tiling of the segmentation into smaller chunks for distributed processing and vectorizes them into GeoPackages (GPKGs). This step can generate a large number of files depending on the segmentation and tile sizes, so folders are created in the input directory to contain the auxillary and final files. Here we do not outline each step in detail and refer the reader to the code for detailed information on each step on the processing chain. This step utilizes Python's multiprocessing package to distribute the processing across multiple ores for efficient processing. This is readily managed for the size of the input dataset provided but is a powerful tool when processing very large segmentations.

To run this step, use:

Python 2_BoundingBoxes_Docker.py -i WV03_20170530_M1BS_104001002EBA0900-toa_ND_k80_n4000_segs_tiles_clump.kea -m WV03_20170530_M1BS_104001002EBA0900-toa_ND_k80_n4000_segs_tiles_clump_modeTileMsk.kea -r 2 -c 4

where:

-i = the input segmentation generated in step 1
-m = the mode tile image generated in step 2
-r = input image resolution
-c = the number of computing cores to use

This will generate a folder named “6_GPKGs” which will contain 147 GPKG files as in Figure 2.

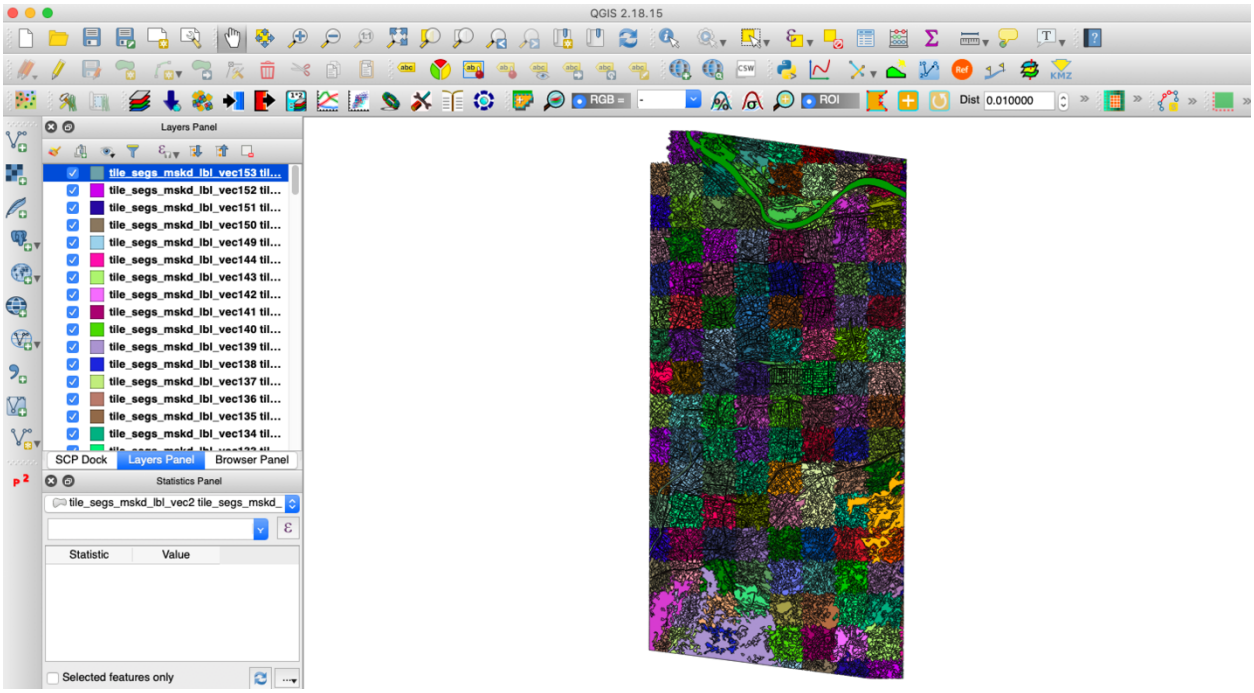

Figure 2: 147 GPKG files displayed in QGIS. Each one is a subsect tile of the input segmentation.

Step 4: Population of the vectors with raster values (3_PopulatePolys.py)

The final step of FARMA is to populate the vector objects with pixel values from input raster data, such as Sentinel-2 time-series NDVI data. For this example, two images are provided for you: 2018 Sentinel-2 mean NDVI images for the months of January and February.

This script will populate the object values with the average NDVI value for each pixel that falls within each object. To include additional statistics, these can be turned on in the script by providing the relevant field name. If the nodata value differs from 0, this should also be specified in the command. For efficiency, this script uses the Python Multiprocessing package but each image band must be provided as a separate image. The name of the column in the attribute table is automatically generated using the last part of the filename. For example for the file “Sentinel2_MonthlyGreenestPixel_2018_NDVI_UTM_Feb.kea” the column name for the mean value is jan_mean. Files should be given appropriate names to enable this.

The code is run as follows:

Python 3_PopulatePolys.py -s ./6_GPKGs -r ./rasters -o ./PopulatedGPKGs -c 4

Where:

-s = the filepath to the GPKG files
-r = the filepath to the input rasters
-o = the output directory
-c = the number of cores to use

The output from this script is 147 new GeoPackage files, each containing the statistics in the attribute table as a new column, as in Figure 3.

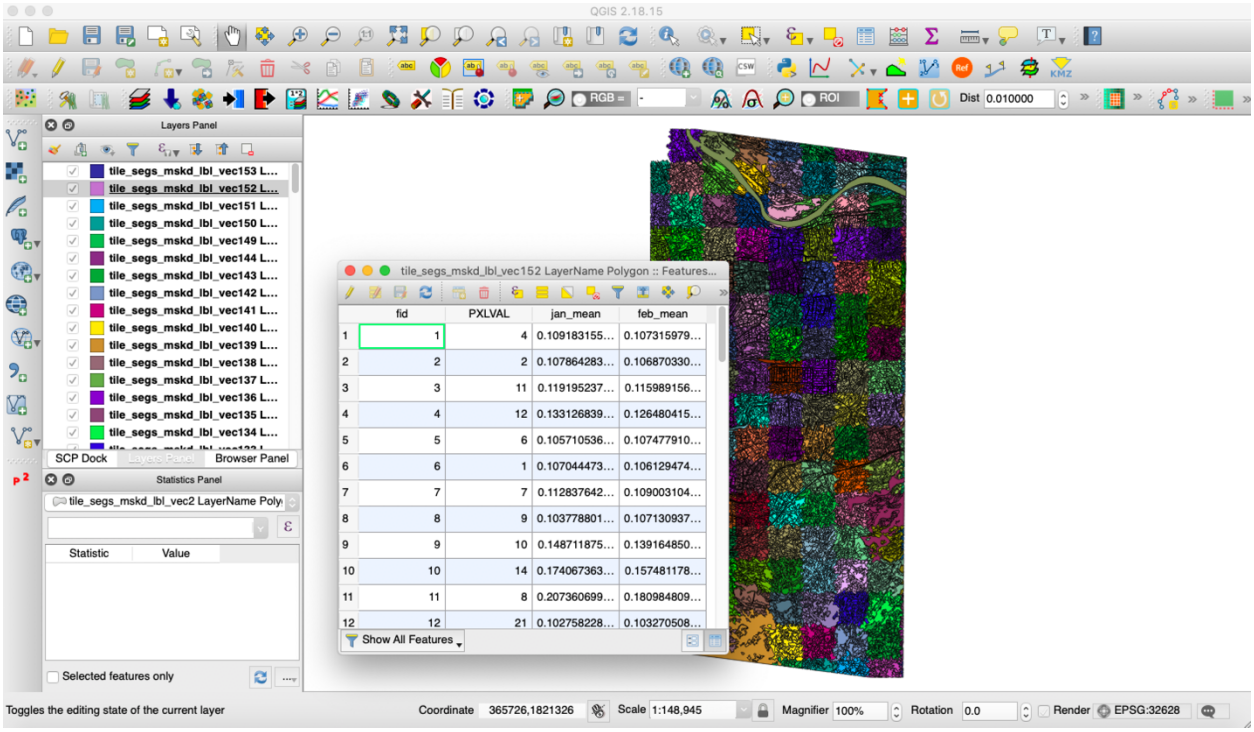

Figure 3. GPKG files with raster values populated into columns of the attribute table