PML | Plymouth Marine Laboratory

EUMETSAT

Ocean Colour Multi-Mission Algorithm Prototype System (OMAPS)

Source Code Installation and Software User Manual

Ref: D 2.6
Date: 14/07/2021
Issue: 1.6
For: EUMETSAT
Ref: ID 995318

Table of Contents

Document Control

Version No.	Release Date	Author/Contributor	Reason for issue
0.1	25/09/2020	Thomas Jackson	Initial draft for structure
1.0	06/10/2020	Thomas Jackson, Olaf Danne, James Dingle, Nicola Martin	Updates for code release to EUMETSAT

1.2	28/03/2021	Thomas Jackson, Olaf Danne, James Dingle, Nicola Martin	Updates following code updates to modules
1.3	14/07/2021	Thomas Jackson, Olaf Danne, James Dingle, Nicola Martin	Updates for v1.3 code release to EUMETSAT

Applicable Documents

ID	Document
AD-1	Eumetsat general Statement of work (requirements from this are given as GR-00 where the number is that given in the reference document)
AD-2	Eumetsat specific Statement of work for ocean colour prototype (requirements from this are given as SOW-00 where the number is that given in the reference document)
AD-3	PML proposal
AD-4	Atmospheric Round Robin Algorithm Theoretical Baseline Document
AD-5	In-water algorithm Round Robin Algorithm Theoretical Baseline Document
AD-6	Requirements Baseline ocean colour processor
AD-7	Requirements Baseline offline processor
AD-8	Input Output Data Definition (IODD)
AD-9	OC-CCI Users Requirements Document (https://esa-oceancolour-cci.org/sites/esa-oceancolour-cci.org/alfresco.php?file=1d95f333-5c78-40be-a25c-8d40d0b161d8&name=Ocean%20Colour%20CCI%20-%20URD%20v4-0.pdf)
AD-10	EUM/SEN3/DOC/19/1092968 Recommendations for Sentinel-3 OLCI Ocean Colour product validations in comparison with in situ measurements – Matchup Protocols

Reference Documents and sources

ID	Document
RD-1	Jackson T., Sathyendranath S., Mélin F., (2017) An improved optical classification scheme for the Ocean Colour Essential Climate Variable and its applications, Remote Sensing of Environment, Volume 203, Pages 152-161, doi:10.1016/j.rse.2017.03.036.
RD-2	Valente, A, Sathyendranath, S, Brotas, V, Groom, S, Grant, M, Taberner, M, Antoine, D, Arnone, R, Balch, W, Barker, K, Barlow, R, Bélanger, S, Berthon, JF, Besiktepe, S, Brando, V, Canuti, E, Chavez, F, Claustre, H, Crout, R, Frouin, R, Garcìa-Soto, C, Gibb, SW, Gould, R, Hooker, S, Kahru, M, Klein, H, Kratzer, S, Loisel, H, McKee, D, Mitchell, BG, Moisan, T, Muller-Karger, F., O'Dowd, L, Ondrusek, M, Poulton, A, Repecaud, M, Smyth, T, Sosik, HM, Twardowski, M, Voss, K, Werdell, J, Wernand, M,

	Zibordi, G (2016) A compilation of global bio-optical in situ data for ocean-colour satellite applications. Earth Syst. Sci. Data, 8, 235–252, 2016www.earthsyst-sci-data.net/8/235/2016/doi:10.5194/essd-8-235-2016
RD-3	Steinmetz F., Mazeran C., Chimot J., (2019) Spectral matching Atmospheric Correction for Sentinel Ocean colour measurements (SACSO) Product Validation Plan.
RD-4	CBIOMES (2019) https://cbiomes.org/research/ocean-color-and-ocean-biogeochemistry
RD-5	AMT (2020) https://www.amt-uk.org/About-AMT
RD-6	LIMNADES (2020) https://limnades.stir.ac.uk/Limnades_login/info_pages/About.php

List of Acronyms

Acronym	Description
ATBD	Algorithm Theoretical Baseline Document
CCI	Climate Change Initiative
ESA	European Space Agency
MERIS	MEdium Resolution Imaging Spectrometer
NASA	National Aeronautics and Space Agency
OC-CCI	Ocean Colour Climate Change Initiative
OLCI	Ocean and Land Colour Instrument
OMAPS	Ocean Colour Multi-Mission Algorithm Prototype System
POLYMER	POLYnomial based algorithm applied to MERIS (though it is now applicable to multiple sensors)
PVP	Product Validation Plan
RBD	Requirements Baseline Document
SOW	Statement Of Work
SVC	System Vicarious Calibration
TOA	Top of atmosphere

1 Purpose

This document is composed of 2 sections. The first section documents information about the software design, location and installation instructions. The second section provides a guide on how to use the software with test demonstrations of the main 'online' processor and the 'offline' support modules.

2 Code Status

2.1 Code location and structure

The codebase is hosted at https://gitlab.eumetsat.int/OC/External/omaps_group/omaps_root (on the EUMETSAT gitlab) and its top-level structure partitions the code into the offline, online, and external dependency components. This complies with the RBD documents [see AD-7] and gives a clear structure for code navigation.

Document last updated for code commit version: 21c286a166cae745d907fb4b1292eaaa54d065c7

2.2 Code Environment

The OMAPS Processing System is designed such that it can be run in a docker environment with python3. This was done so that users do not need to set up a virtual-machine (vm) themselves to comply with the processor requirements (such as python libraries etc) which should increase the ease of setup and creation of parallel environments. We have included the initial docker build here (along with simple test instructions) with a number of tests and we also give an example of using 'singularity' to run docker style processing on a grid environment in which docker root permission is not available.

Where code snippets are given they use the unix directory notation with folders separated by '/'. This is done because the code was designed to run on a docker and the docker environment is unix. Most of the code will work outside of the docker on a linux or Mac OS as these are also unix based systems. The docker should work on windows systems but windows users will face significant issues if they try to run some of the python code outside of the docker environment due to factors such as windows folder notation.

2.3 Required functionality of the OMAPS Processing System

The OMAPS Processing System is composed of an online and an offline processor such that it can be used for both operational data processing with a fixed configuration or for the comparison of the performance of differing configuration options.

One example of the offline processor functions is matchup extraction. A graphical outline of the components required for a fully functional matchup demonstration are shown in Figure 1. Most of the components are complete and available in the code repository (dark green). As these components are required for the SVC processing that component is shaded light green, though the code itself is operational. This is currently under investigation and is the reason that two boxes are shaded yellow in figure 1. For demonstration purposes we have identified a granule that matches an in-situ measurement (as would be required for matchup analysis or SVC calculation) and tested all other code elements with this. The remaining three pale green boxes are all components that are under development:

1) The config format for the online processor is simple to interpret and modify (see section 4.4 for an example of updating the config) but we are not happy to call this complete yet as we want to improve the ability of the offline processor to tailor the online config (setting water class blending scheme for example).

2) The Environment/configuration checker function will now effectively be performed through the use of docker.
3) The matchup data base (MDB)format handed back to the SVC should be as expected, given the tests we have performed with the matchup module but as we cannot do the end to end SVC test (due to the OCDB issues) we are not yet willing to call this component completely tested.

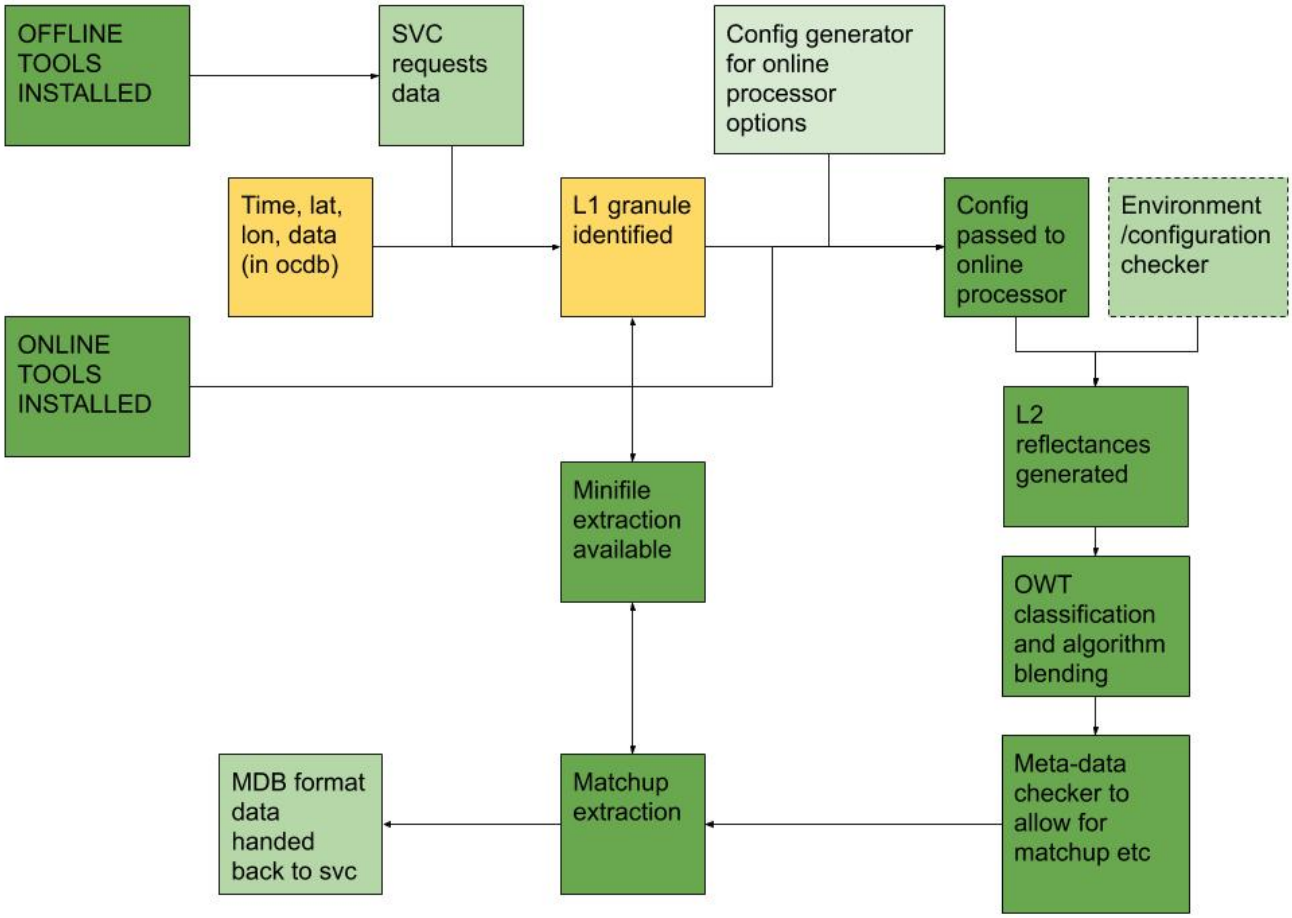

Figure 1: Processing diagram showing all the components required and order of operation for processing data as part of an SVC calculation.

All progress and issues relating to the codebase are monitored through the gitlab system. This system allows developers and reviewers to be tagged in relation to issues, clear tracking from opening to closing of issues, simultaneous development of features on different code branches before merging into the primary branch.

2.4 Processing requirements (hardware)

The OMAPS Processor System can be deployed in a parallel environment (such as grid-based processing) as well as run on a single machine. Some processing (such as IDEPIX classification) uses as many cores as are available on a given machine and others (such as POLYMER) can be configured to use a fixed number of cores. The code has been tested on a virtual machine with access to 32GB RAM and 4 cores for processing. With these specifications the processing of a complete full-resolution OLCI granule took approximately 25-30 minutes. We have also performed the same operation on a much more powerful machine (32 Cores and 64GB of RAM available) which took <10 minutes to process the same granule (maxing out at 25GB RAM used on 16 cores during POLYMER processing).

It is known that the SACSO atmospheric correction processor is currently limited to making use of a single core for processing. This means that processing a single granule using the SACSO processor takes 2-4 times longer than using POLYMER but if the jobs are run in parallel such that multiple jobs are using single cores

then the overall loss of efficiency can be somewhat mitigated. Though this can put higher RAM requirements on the processing machine.

3 Software setup

Below is a step by step guide to installing the software from the EUMETSAT gitlab repository.

3.1 Installation

If not already available, install 'git' for the given Linux or mac OS distribution (see https://git-scm.com/download/linux).

Also ensure that docker is installed on the machine in question (https://docs.docker.com/get-docker/).

Before starting the installation it is recommended to choose a suitable installation location and create and OMAPS root directory into which the code will be installed. We will refer to this top level directory as <OMAPS_ROOT>. You will then need to move to this directory and set the environment variable for OMAPS_ROOT.

```
mkdir OMAPS
cd OMAPS
export OMAPS_ROOT=$(pwd)
```

You will also need to ensure that you have setup a login to access the EUMETSAT gitlab with an sshkey. Then you can run the following from the OMAPS_ROOT directory.

```
git clone
https://gitlab.eumetsat.int/OC/External/omaps_group/omaps_root.git
```

This should have created an ‘omaps_root’ directory in the OMAPS_ROOT folder that contains a suite of directories such as online_processor, offline_processor and external_dependencies. We will now build the docker image and ensure that the online and offline processors are set up correctly.

```
cd $OMAPS_ROOT/omaps_root
git checkout -b dev origin/dev
sudo docker build -t omaps/omaps_processor --build-arg USER_ID=$(id -u)
--build-arg GROUP_ID=$(id -g) --build-arg BUILD_OS=linux-gnu .
```

If not building the docker on a linux machine (such as a mac OS) then you can remove the “--build-arg BUILD_OS=linux-gnu”.

Note above that sudo rights are required to build the docker image. Sudo rights are not required to use the docker so if you do not have sudo permissions you may have to get a more senior user to build an image for you. This docker image building will download and install a number of programs such as SNAP and POLYMER. For those unfamiliar with docker, the last line above should have created a docker image (like a virtual machine) with the name omaps/omaps_processor. If you want to see what software and packages are installed on the docker image then you can find the contents in <OMAPS_ROOT>/omaps_root/Dockerfile.

To use the docker within EUMETSAT bulk processing environment, push it into harbour:

```
docker login harbor.opscloud.eumetsat.int  (require eumetsat credential)
```

```
docker tag omaps/omaps_processor harbor.opscloud.eumetsat.int/rsp-
s3/omaps_processor
docker push harbor.opscloud.eumetsat.int/rsp-s3/omaps_processor
```

3.1.1 The OMAPS conda environment

Part of the docker build process creates an 'anaconda' python environment within the docker called 'omaps'. If you want to have a duplicate version of this environment on your machine (accessible outside the docker) then run the following commands:

```
cd $OMAPS_ROOT/omaps_root
cp omaps_environment.yml omaps_environment_outside_docker.yml
sed -i 's@/opt/omaps/@'"$OMAPS_ROOT/omaps_root/"'@'
omaps_environment_outside_docker.yml
```

NOTE: on Mac OS you will need to use the following line instead of the one above

```
sed -i '' 's@/opt/omaps/@'"$OMAPS_ROOT/omaps_root/"'@'
omaps_environment_outside_docker.yml

conda env create --file omaps_environment_outside_docker.yml
```

You can now activate the environment outside of the docker with:

```
conda activate omaps
```

If you update the codebase at any point (likely through a git pull to add updates from the central codebase) you will need to rebuild the docker image so that the code within the docker also has the updates. You would also need to rebuild the docker if you checkout a different branch of the codebase (though in that case you might change the name of the docker image you build to reflect the different branch). Other than for code update the docker image should not need to be rebuilt and should run on any machine with docker available. It can therefore be built on one machine and then the image can be export to or sourced by other machines.

3.2 Test docker works

You can run docker images in two modes. The first option is to run the docker interactively, where you can run commands as if running on a virtual machine. The second option is to submit predefined jobs to the docker for processing. We will cover both these types of interaction below.

In order to run the docker tests we will need a test working directory containing an OLCI level1 file. You may have a file already but if not then you can download one from the EUMETSAT CODA archive or use the following commands to create an example workspace in your /tmp/ space

```
cd /tmp/
#Download the file from webhost
wget -nH --cut-dirs=5 -r -R "index.html*" --no-parent -e robots=off
https://rsg.pml.ac.uk/shared_files/OMAPS/test_data/workspace_S3A_OL_1_EF
R____20170522T090453_20170522T090653_20171018T123841_0119_018_050______M
R1_R_NT_002.SEN3/0_initial_state/
#Make the workspace structure that the OMAPS system expects
mkdir -p workspace_S3A_OL_1_EFR____20170522T090453_20170522T090653_20171
018T123841_0119_018_050______MR1_R_NT_002.SEN3/0_initial_state/
#put the data folder into the '0_initial_state' folder
mv S3A_OL_1_EFR____20170522T090453_20170522T090653_20171018T123841_0119_
018_050______MR1_R_NT_002.SEN3 workspace_S3A_OL_1_EFR____20170522T090453
```

```
_20170522T090653_20171018T123841_0119_018_050______MR1_R_NT_002.SEN3/0_i
nitial_state/
```

If you do wish to create your own workspace for use with the online processor you need to ensure that it has the same structure as this example workspace. The structure in the workspace is <workspace_name>/0_initial_state/<OLCI_level1.SEN3 file> which looks like this when displayed with the 'tree' command:

```
workspace_S3A_OL_1_EFR____20170522T090453_20170522T090653_20171018T12384
1_0119_018_050______MR1_R_NT_002.SEN3
└── 0_initial_state
    └──S3A_OL_1_EFR____20170522T090453_20170522T090653_20171018T123841_
0119_018_050______MR1_R_NT_002.SEN3
        ├── Oa01_radiance.nc
        ├── Oa02_radiance.nc
        ├── Oa03_radiance.nc
        ├── Oa04_radiance.nc
        ├── Oa05_radiance.nc
        ├── Oa06_radiance.nc
        ├── Oa07_radiance.nc
        ├── Oa08_radiance.nc
        ├── Oa09_radiance.nc
        ├── Oa10_radiance.nc
        ├── Oa11_radiance.nc
        ├── Oa12_radiance.nc
        ├── Oa13_radiance.nc
        ├── Oa14_radiance.nc
        ├── Oa15_radiance.nc
        ├── Oa16_radiance.nc
        ├── Oa17_radiance.nc
        ├── Oa18_radiance.nc
        ├── Oa19_radiance.nc
        ├── Oa20_radiance.nc
        ├── Oa21_radiance.nc
        ├── geo_coordinates.nc
        ├── instrument_data.nc
        ├── qualityFlags.nc
        ├── removed_pixels.nc
        ├── tie_geo_coordinates.nc
        ├── tie_geometries.nc
        ├── tie_meteo.nc
        ├── time_coordinates.nc
        └── xfdumanifest.xml
```

3.2.1 Interactive Docker

To run the docker image and spin up the virtual machine simply run:

```
python3 $OMAPS_ROOT/omaps_root/run_docker_container.py <workdir>
```

where the <workdir> is a directory that you wish to be mounted on the virtual machine so that it's contents are visible to the processor (this will usually be a directory full of data).

You should replace <workdir> with a local directory that contains your OLCI data in the 0_initial_state sub directory as specified above. For the example above this would be /tmp/ workspace_S3A_OL_1_EFR____20170522T090453_20170522T090653_20171018T123841_0119_018_050_____MR1_R_NT_002.SEN3 . The workdir path need to be an absolute path rather than a relative path.

It is important to note that the docker may run into write permission errors if you try to mount a workspace that is network storage on an NFS server. If the data that you wish to process is hosted on such a network storage then you will need to copy it to the local hard-drive (which will also increase read-write speeds for processing) and then move the processed data back to network storage outside the docker once processing is completed. An example of doing this as part of a bulk data processing operation is given in section 4.5.

If you wish to use the GUI's available in some of the offline processor modules then you will need to specify 'x-forwarding' (which may only be available with sudo privileges) by running:

```
python3 run_docker_container.py <workdir> -x
```

This command (with or without x forwarding) will open an interactive session inside the container, where you will be able to run simple bash commands in the OMAPS environment, and run your choice of scripts from the online_processing and offline_processing directories.

If running interactively, the online_processor and offline_processor code can be found in your current directory (/opt/omaps/) and the data directory will be mounted at the top level (/workspace/).

You can close the interactive session with 'ctrl-d' or by typing:

```
exit
```

3.2.2 Predefined docker operations

There are a number of predefined operations available. These are defined within the 'container_commands.py' file. One such predefined operation is an integration test which will perform a test run of the online processor to check the installation is correct. Run the online processing integration tests directly with:

```
python3 run_docker_container.py <workdir> -r integration_tests
```

This will run the tests in the container (in the background) and save results to your <workdir>.
Once dispatched you can check if it is running with:

```
docker ps
```

You can stop the tests from running at any time with:

```
docker stop omaps
```

To view the logs of your running container you can run (exit the log view with ctrl-c)

```
docker logs -f omaps
```

Once this processing is complete you should see all the stages populates with output products in the <workdir> directory (the processing will have been undertaken using the default processor, POLYMER).

Similarly the offline processor configuration tool can be run with:

```
python3 run_docker_container.py <workdir> -r config_tool
```

This will open the configuration tool GUI, where you will be able to create and save your .ini file. Save this to the <workdir> if you want to access it outside of the container.

When you exit the GUI, you will also exit the docker container.

4 Module tests

4.1 Test data

As with the integration test example above we will focus on a single granule of OLCI 3A data where possible for initial testing (S3A_OL_1_EFR____20170522T090453_20170522T090653_20171018T123841_0119_018_050_____MR1_R_NT_002.SEN3). This granule is an image taken over the Mediterranean (the region around Sicily) on the 22nd of May 2017. This granule was chosen as:

- It has a large amount of cloud free open water in the image.
- It covers a range of environments from coastal to low chlorophyll-a waters.
- It contains clouds to test cloud masks and associated flags (cloud buffers, cloud shadows etc)
- Contains some glint to allow testing of polymer in glint conditions, but is not dominated by glint.
- It overlaps with an in-situ measurement that is present in the OCDB database (taken during the PEACETIME cruise).
- It is sufficiently old that the calibration and processing version is likely stable.

Figure 2: True colour image for the test granule used. The matchup location is shown by pin 1.

We will use this file to demonstrate the creation of a processing workspace, conversion from L1 to L2 including optical classification and algorithm blending, and matchup extraction. Running a minifile through the processor would make the chain through to extract matchup much faster but the minifile is not currently compatible with polymer. We will address this moving forward to allow a large speedup in matchup processing.

All demonstrations below assume that you have followed the installation instruction in section 3.1 and are running the tests on the virtual machine.

4.2 Using the Granule_Finder

Within the offline processor system there is a granule finder which can be used to search for granules using an API call to CODArep and CODA data repositories. This is done by supplying a config file with a spatial and temporal region of interest. More details on creating configuration files is given in section 4.6.1. Using a pre-made configuration file, the following examples can be performed within the docker (first example below) or outside the docker (second example below, noting that you must have a suitable python environment set).

Example Granule Query using the docker system:

```
cd $OMAPS_ROOT/omaps_root/
python3 run_docker_container.py $(pwd)
python ./offline_processor/preparation_olci_granule_finder.py --
configfile
./offline_processor/test/demo_configs/preparation_olci_granule_finder_te
st_Med.ini
```

Example Granule Query using using native python:

```
conda activate omaps
```
(See section3.1 for creation of omaps conda environment)

```
cd $OMAPS_ROOT/omaps_root/
python ./offline_processor/preparation_olci_granule_finder.py --
configfile
./offline_processor/test/demo_configs/OLCI_granule_finder_test_Med.ini
```

This should print the following filename to the terminal:

```
1 files found in https://codarep.eumetsat.int
S3A_OL_1_EFR____20170522T072940_20170522T073140_20171018T123546_0119_018
_049______MR1_R_NT_002.SEN3
```

There will also be a file written to /tmp/ (as this is specified as the output location in the config file) called OLCI_granules.txt that contains all the matching files (1 in this case). Don't forget that if you want this file to be visible once you close the docker you will need to move it to /workspace/ as this is the mounted directory from the primary machine).

Note: There currently exists a temporal gap in the CODARep + CODA database as CODA provides granules from the last year only and CODARep has not been updated recently.

4.3 Test Workspace creation

First, we will create a general location for workspaces and then create a particular test workspace containing the S3A_OLCI file of interest. The name 0_initial_state is the name that will be recognised by the processing supervisor when we pass it the workspace so should not be modified. The processing supervisor is part of the online processor subsystem that checks what stages of processing have been completed and which are yet to be completed.

We have created a python script ($OMAPS_ROOT/omaps_root/ancillary_tools/workspace_creator.py) that will create a correctly formatted workspace in a location of your choice (below we specify the /tmp/ location) given 1) a data directory in which to search for an olci file, 2) a regex string that will allow unique identification of the granule of interest and 3) an output location.
As an example, if the granule was available somewhere in the directory /data/datasets/OLCI/level1/ the workspace creator command would be:

```
python $OMAPS_ROOT/omaps_root/ancillary_tools/workspace_creator.py -dds
/data/datasets/OLCI/level1/ -fs S3A_OL_1_EFR____20170522T090453_ -o
/tmp/
```

You will need to update the '/data/datasets/OLCI/level1/' above to wherever you are storing your level 1 OLCI data.

To save time on the search for a matching file it helps if you can provide as much detail on the data directory as possible if you are pointing at a very large archive, such as:

```
python $OMAPS_ROOT/omaps_root/ancillary_tools/workspace_creator.py -dds
/data/datasets/OLCI/level1/2017/05/22/ -fs
S3A_OL_1_EFR____20170522T09045 -o /tmp/
```

where this might limit the search to granules across a single day.

4.4 L1 to L2 processing

With your chosen L1B granule in the 0_initial_state directory of your workspace, running the current online processor configuration is just a case of running the **run_workspace_config.py** file in the online_processor directory.
This can either be done through a docker call or using the code directly on the current machine.
This script requires two arguments: the path of the workspace (containing the 0_initial_state folder) and the location of the .ini config to process. Multiple example configuration files are available following installation and users can also create their own configuration files. For the following test we will use a POLYMER configuration file with IdePiX masking turned on (relative path: *online_processor/test/demo_configs/polymer_demo.ini*)
In the interest of speed, the cloud shadow computation in Idepix has been turned off in the test config (approximately 3-5x slower). But should you want to test with the full capability of Idepix at this stage, it can be re-enabled by modifying the config .ini file.
Specifically, by modifying the command line in the first stage (Idepix) and changing the flag -PcomputeCloudShadow=False to True.
Currently the section in the polymer_demo.ini titled 'path_replacements' includes the following:

```
omaps_root=/opt/omaps
polymer=/opt/omaps/external/polymer-v4.13
polymer_sacso=/opt/omaps/external/polymer-sacso-v0.9
snap_install_dir=/opt/omaps/external/
omaps_conda_env=/opt/miniconda/envs/omaps/bin/python
polymer_conda_env=/opt/miniconda/envs/omaps/bin/python
```

4.4.1 Running in interactive docker

Begin by initialising the docker with your workspace mounted:

```
cd $OMAPS_ROOT/omaps_root
./run_docker_container.py
/tmp/workspace_S3A_OL_1_EFR____20170522T090453_20170522T090653_20171018T
123841_0119_018_050______MR1_R_NT_002.SEN3
```

Then you can run the online processor on the test granule using the following command:

```
python ./online_processor/run_workspace_config.py --config
./online_processor/test/demo_configs/polymer_demo.ini --workspace
/workspace/
```

Once this is complete you should find all various processing stages available in /workspace/ on the docker (and in the workspace folder location that you specified to the docker on initialisation when you exit the docker).

It is also of note that if you do not want to dig into any of the intermediate products after using the test processor config then it is worth while to delete the stages 0 to 8 and only retaining the final stage output file in order to safe on storage space. This can be done with a simple:

```
rm -rf /tmp/<workspace>/[0-8]*
```

4.4.2 Running as predefined docker operation

In order to use the predefined docker operation (run by the docker in the background without initialising first) you can use the 'run_docker_container.py' command with the -r flag to specify the operation. For example to process a workspace from L1 to L2 using polymer one could run:

```
cd $OMAPS_ROOT/omaps_root
python run_docker_container.py <workdir> -r polymer_demo
```

If the above command starts up a docker container correctly then it will return a long, seemingly random, string which is the id of the docker instance. For example
ea012573a6a2a3acabcb96eba718e8fa038d4b9524154997b8656b3b78e046b2

If you run the docker ps command then it should return something similar to below and you can see the 'Container ID Name' will match the string.

```
CONTAINER ID  IMAGE                   COMMAND                  CREATED         STATUS         PORTS     NAMES
ea012573a6a2  omaps/omaps_processor   "/bin/bash -c 'pytho…"   4 seconds ago   Up 3 seconds             omaps
```

A complete list of current predefined docker operations is in $OMAPS_ROOT/omaps_root/container_commands.py.

To process a workspace with the sacso configuration for example one could run:

```
cd $OMAPS_ROOT/omaps_root
python run_docker_container.py <workdir> -r sacso_demo
```

To process a workspace with the l2gen configuration for example one could run:

```
cd $OMAPS_ROOT/omaps_root
python run_docker_container.py <workdir> -r l2gen_demo
```

You could also add new commands to this list if you have another operation you wish to repeat using the omaps docker environment.

4.4.3 Running outside the docker

In order to run the processing outside of the docker you will need to ensure that your main environment has all the packages and programs that are installed on the docker image. If you wish to do this then you can find the list of package and programs installed on the docker in $OMAPS_ROOT/omaps_root/Dockerfile. This would include the creation of the omaps conda environment.

As mentioned in the installation guide, you can create a conda environment to match the one on the docker by using the file 'omaps_environment.yml' in the omaps_root folder:

```
cd $OMAPS_ROOT/omaps_root
cp omaps_environment.yml omaps_environment_outside_docker.yml
```

```
sed -i 's@/opt/omaps/@'"$OMAPS_ROOT/omaps_root/"'@'
omaps_environment_outside_docker.yml
```

NOTE: on Mac OS you will need to use the following line instead of the one above

```
sed -i '' 's@/opt/omaps/@'"$OMAPS_ROOT/omaps_root/"'@'
omaps_environment_outside_docker.yml

conda env create --file omaps_environment_outside_docker.yml
```

To run the online_processor outside of the docker you will also need to create a config file in which these paths pointed where you have installed the omaps_root directory e.g:

```
omaps_root=<your $OMAPS_ROOT>/omaps_root
polymer=<your $OMAPS_ROOT>/omaps_root/external/polymer-v4.13
snap_install_dir=<your $OMAPS_ROOT>/omaps_root/external/
omaps_conda_env=/home/<your user space>/anaconda3/envs/omaps/bin/py
thon
polymer_conda_env=/home/<your user space>/anaconda3/envs/polymer/bi
n/python
```

If you had placed this modified config file (your_modified_config.ini) in /tmp/ then the example L1 to L2 command to process
data in /tmp/workspace_S3A_OL_1_EFR____20170522T090453_20170522T090653_20171018T123841_0119_018_050_____MR1_R_NT_002.SEN3/ would then be:

```
conda activate omaps
python $OMAPS_ROOT/omaps_root/online_processor/run_workspace_config.py -
-workspace
/tmp/workspace_S3A_OL_1_EFR____20170522T090453_20170522T090653_20171018T
123841_0119_018_050_____MR1_R_NT_002.SEN3/ --config
/tmp/<your_modified_config.ini>
```

4.5 Using Singularity

In some processing environments it is possible to run multiple instances of the same docker image so that you could dispatch a whole suite of granule processing jobs for completion. In some environments however the root permissions required by docker can mean that massive grid processing operations may not be possible using docker. In this case we turn to singularity to allow the grid processing of many hundreds or thousands of granules from a singularity image. Using the singularity image is extremely similar to using the docker image but it doesn't require the same level of user privileges. An example of using singularity for massive parallel processing is given below.

Firstly we must create a singularity image from the docker image. This has to be done on a local machine rather than a networked storage space (in the example below we write to 'scratch_local'. Once the build has completed you can then move the singularity image (.sif file) to a network visible location for many machines to source (in the example below we move it to 'scratch_network').

So with the docker image built in $OMAPS_ROOT/omaps_root:

```
cd $OMAPS_ROOT/omaps_root
sudo singularity build ~/scratch_local/omaps.sif docker-
daemon://omaps/omaps_processor:latest
mv ~/scratch_local/omaps.sif ~/scratch_network
```

For more details on passing commands to the singularity image please see https://singularity.hpcng.org/user-docs/master/quick_start.html#interact-with-images. An example below shows how to use the singularity image (in the same way you would use the docker image) to process a granule using the default polymer demo config. The similarity between this command and those in 4.4.1 and 4.4.2 should be clear.

```
singularity run --bind $ws:/workspace --bind /tmp --containall
~/scratch_network/omaps.sif \
/bin/bash -c ". activate omaps; python
/opt/omaps/online_processor/run_workspace_config.py \
--config /opt/omaps/online_processor/test/demo_configs/polymer_demo.ini \
--workspace /workspace"
```

4.5.1 Working with data on network storage

As mentioned in section 3.2.1 the docker will only have write permissions to local storage, not NFS storage locations. This means that for batch processing the workspace will need to be created on the local machine doing the processing and run there. This has some advantages as the access speed of the local storage will likely be higher than the network storage. This means that the read and writes for all the intermediate stages of the OMAPS processor should benefit.

In the following example script jobs are processed sourcing data from a network location and writing out to a different network storage location with intermediate products kept local. At the end of the processing intermediate stages are removed to reduce the final output volume and the products are moved from local to network storage.

```
cd $OMAPS_ROOT/omaps_root
./ancillary_tools/network_storage_file_process_example.sh
<network_folder_with_input_options> <input_regex> <output_directory>
<singularity_docker_sif_location>
```

For example if you were looking to process a file matching 'S3B_OL_1_EFR____20201010T102642_*' somewhere in the dir '/data/sentinel3b_olci/level1/non_time_critical/swath/0d/2020/10/10/' and outputting to '~/scratch_network/OMAPS_out' using the singularity instance created above you could run:

```
./ancillary_tools/network_storage_file_process_example.sh
/data/sentinel3b_olci/level1/non_time_critical/swath/0d/2020/10/10/
S3B_OL_1_EFR____20201010T102642_* ~/scratch_network/OMAPS_out/ ~/scratch
_network/omaps.sif
```

A EUMETSAT specific example of a batch processing script is given at:

```
$OMAPS_ROOT/omaps_root/ancillary_tools/eumetsat-
bulk_run_omaps_online_processor.sh
```

4.6 Matchups

The purpose of the MATCHUP module is to create a matchup data base (MDB) by comparing the contents of the OCDB to a set of Ocean Colour granules which need to be generated as described in detail in section 4.4. In the current section, the sequence of steps to create a MDB is outlined. To illustrate the procedure, a minimal test case of the generation of a MDB between a single granule and data within the OCDB is demonstrated. However, typical use cases for the generation of a reasonable number of matchups will of course require a much larger number of granules.

4.6.1 Find OLCI L1 granules for a defined matchup region and time interval

For the creation of a matchup database, a specific region and time interval needs to be defined from which all available OLCI L1 granules from the Eumetsat CODA repository can be, and in ideal case should be, used as satellite data input. To find out the available granules, the OMAPS Offline processor provides an 'OLCI granule finder' tool, which is basically a Python script writing a list of such products for a given configuration (i.e. region and time interval specified in a configuration ini file).

4.6.1.1 Configuration file

A configuration file for the OLCI granule finder must be provided in ini format and contain the parameters listed in Table 4-1.

Table 4-1: OLCI granule finder config file

PARAMETER	VALUE(S)	DESCRIPTION
coda_query_region	lat_min,lat_max, lon_min,lon_max	Bounding box with minimum/maximum latitudes/longitudes
coda_query_start_date	yyyyMMdd	Start date of matchup time interval
coda_query_start_time	hh:mm:ss	Start time of matchup time interval
coda_query_end_date	yyyyMMdd	End date of matchup time interval
coda_query_end_time	hh:mm:ss	End timeof matchup time interval
output_dir	/path/to/dir	List of files is written into a text file <output_dir>/OLCI_granules.txt

An example file is shown in Figure 3.

```
[OLCI_GRANULE_FINDER]
; configuration for finding olci granules
coda_query_region = -125.0,-117.0, 30.0,35.0
coda_query_start_date = 20160101
coda_query_start_time = 00:00:00
coda_query_end_date = 20170120
coda_query_end_time = 23:59:59
output_dir = /home/olafd/omaps_test/find_olci_granules/output
```

Figure 3: Example for an OLCI granule finder config file

4.6.1.2 Configuration GUI

The config file is basically a simple text file which can be edited manually. However, it is more convenient and less error prone to use the OMAPS configuration tool:

```
# From $OMAPS_ROOT as specified in section 3.1, go to the
# Offline processor subdirectory:
cd $OMAPS_ROOT/omaps_root/offline_processor

# Start the GUI:
./start_offl_proc_config.bash
```

The main screen of the configuration tool should appear. From the menu, the entries ‘Module → PREPARATION → Find OLCI Granules’ must be selected (Figure 4), then the configuration table should appear. The necessary entries can be made here and can be saved (and reloaded if needed) via the ‘File’ menu (Figure 6).

NOTE: On Macintosh systems the menu for the GUI is in the mac-menubar (top left of desktop window) rather than contained in the popup window. This is normal behaviour for Mac systems but may trip up new mac users (See Figure 5).

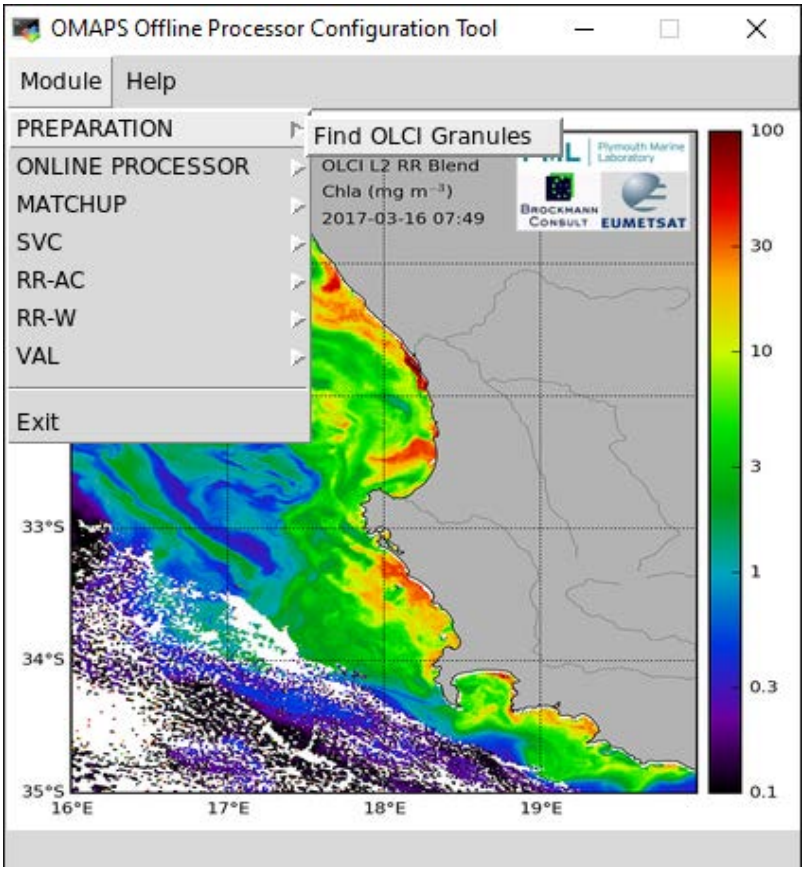

Figure 4: OMAPS configuration tool: Menu to enter configuration for OLCI granule finder

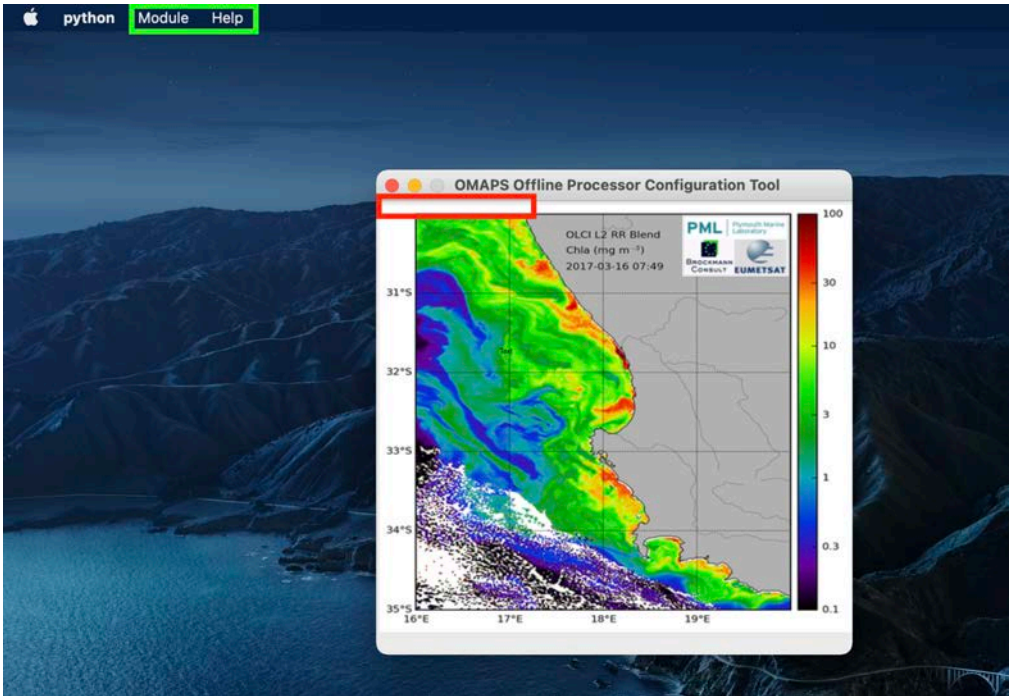

Figure 5: OMAPS configuration tool on Mac OS: Menu location outlined in green & equivalent location for linux (red).

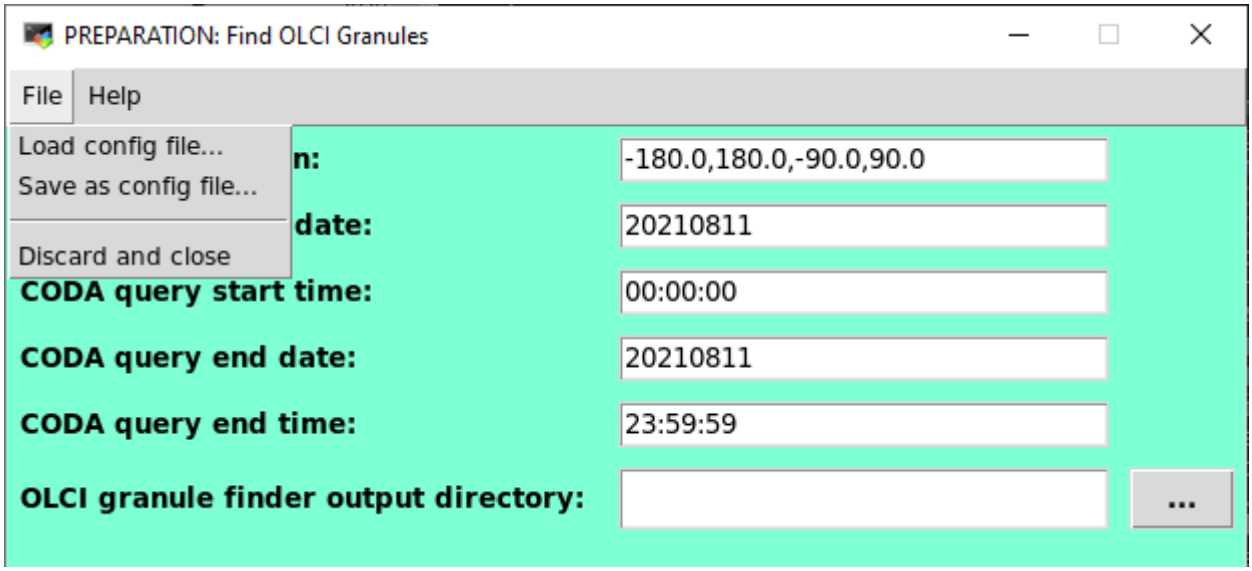

Figure 6: OMAPS configuration tool: Configuration for OLCI granule finder

4.6.1.3 Run OLCI granule finder

After generation of the config file, the OLCI granule finder can be started from a bash script:

```
# From $OMAPS_ROOT as specified in section 3.1, go to the
# Offline processor subdirectory:
cd $OMAPS_ROOT/omaps_root/offline_processor

# Start the OLCI granule finder:
./PREPARATION_find_olci_granules.bash --configfile=<path/to/config>
```

The OLCI granule finder will make queries into the Eumetsat CODA repository and search for OLCI L1 products *S3A_OL_1_EFR____*.SEN3*. The list of products found is written into a text file <output_dir>/OLCI_granules.txt.

4.6.2 Download of L1 granules

If not already available and accessible from the current processing system, the L1 granules as found in the previous step need to be downloaded from the Eumetsat CODA repository with appropriate tools, such as wget. Note that the MATCHUP module will expect L1 granules for L1 matchups in a directory tree like <L1 granules root directory>/yyyy/MM/dd/S3*_OL_1_E*.SEN3. Thus, the products need to be downloaded accordingly, or appropriate symbolic links need to be set. We have created a script for creating suitable directory structures ($OMAPS_ROOT/omaps_root/ancillary_tools/create_chrono_dir_struct_symlinks.sh). An example using this script will be covered below.

4.6.3 Processing L1 to L2

The ‘L1 to L2’ step, provided by the OMAPS Online processor, has been described in detail in section 4.4. It needs to be applied on every granule which was downloaded and made accessible in the previous step. If OLCI standard L2 products generated by the IPF processor shall be used for the matchup generation, no

action from the Online processor is required, but the IPF products need to be downloaded and made accessible for the MATCHUP module.

4.6.4 Matchup generation outside the docker

This section illustrates the basic steps of the generation of a MDB with the OMAPS Offline processor outside a docker environment. This assumes that all prerequisites (such as required Python libraries etc.) are available on the given system where the software has been installed. The omaps conda environment (creation outside of the docker is covered in section 3.1.1) should provide all the required python packages.

4.6.4.1 Matchup configuration

The generation of a MDB using the Offline processor MATCHUP module must also be configured by a configuration ini file. The file format is the same as described above for the OLCI granule finder, and it is again recommended to generate the file with the OMAPS configuration tool.

IMPORTANT: If you intend to use the matchups for use in a round robin exercise then you must only request the bands you wish to compare in the round robin comparison and this band set must be the same for all of the atmospheric processor options you wish to compare. Also note that it is important to check that the waterclass file and whether the water classes are normalised is specified correctly (the same as you would have used for the processing) in the matchup config otherwise you may find discrepancies between the minifiles and the fully processed granules.

The tool is started as described above for the OLCI granule finder:

```
# From $OMAPS_ROOT as specified in section 3.1, go to the
# Offline processor subdirectory:
cd $OMAPS_ROOT/omaps_root/offline_processor
# Start the GUI:
./start_offl_proc_config.bash
```

The main screen of the configuration tool should appear. From the menu, the entries ‘Module → MATCHUP → Configure MDB generation’ must now be selected, then the configuration table should appear. Again, the necessary entries can be made here and can be saved (and reloaded if needed) via the ‘File’ menu (Figure 6).

As we can see from Figure 7, there are lots of configuration parameters. A complete table with possible values and descriptions of all these parameters is given in the OMAPS IODD [AD-8], together with an example of a configuration ini file. In the configuration GUI (Figure 6), a short description for each parameter is provided with tooltips.

However, some important notes for the MATCHUP module configuration should also be summarized here:

- The parameters ‘OCDB username’ and ‘OCDB password’ must not be empty. A user who wants to use the OMAPS MATCHUP module needs to have a valid OCDB account, provided by Eumetsat.
- Matchups can be generated using L1, or L2 products (OLCI L2 standard (IPF processor), or products generated by the OMAPS Online processor). Here, the supported Level-2 processors are POLYMER, SACSO and L2GEN. The L1 product matchups are used by the SVC module.

- MATCHUP module expects satellite input products (L1 or L2) in a directory tree like <satellite input directory>/yyyy/MM/dd
- The parameter ‘Label for this production’ must not be empty, as it is used as subdirectory name for the storage of intermediate data (i.e. satellite micropixel extraction files).
- The option ‘Apply BRDF normalisation’ should be selected for matchups from IPF, but should not be selected for matchups from POLYMER, SACSO or L2GEN (under default processing). It has no effect for matchups from L1 products.
- For the ‘OCDB parameters’ (the insitu/satellite matchup variables), the instructions and rules described in the IODD [AD-8] need to be followed carefully.
- The parameter ‘Valid expression’ refers to the validity of satellite pixels. It is expected in Python syntax and should refer to variables present in the satellite product. If the field is left empty, the default expression for the given processor (see IODD, [AD-8]) is used.

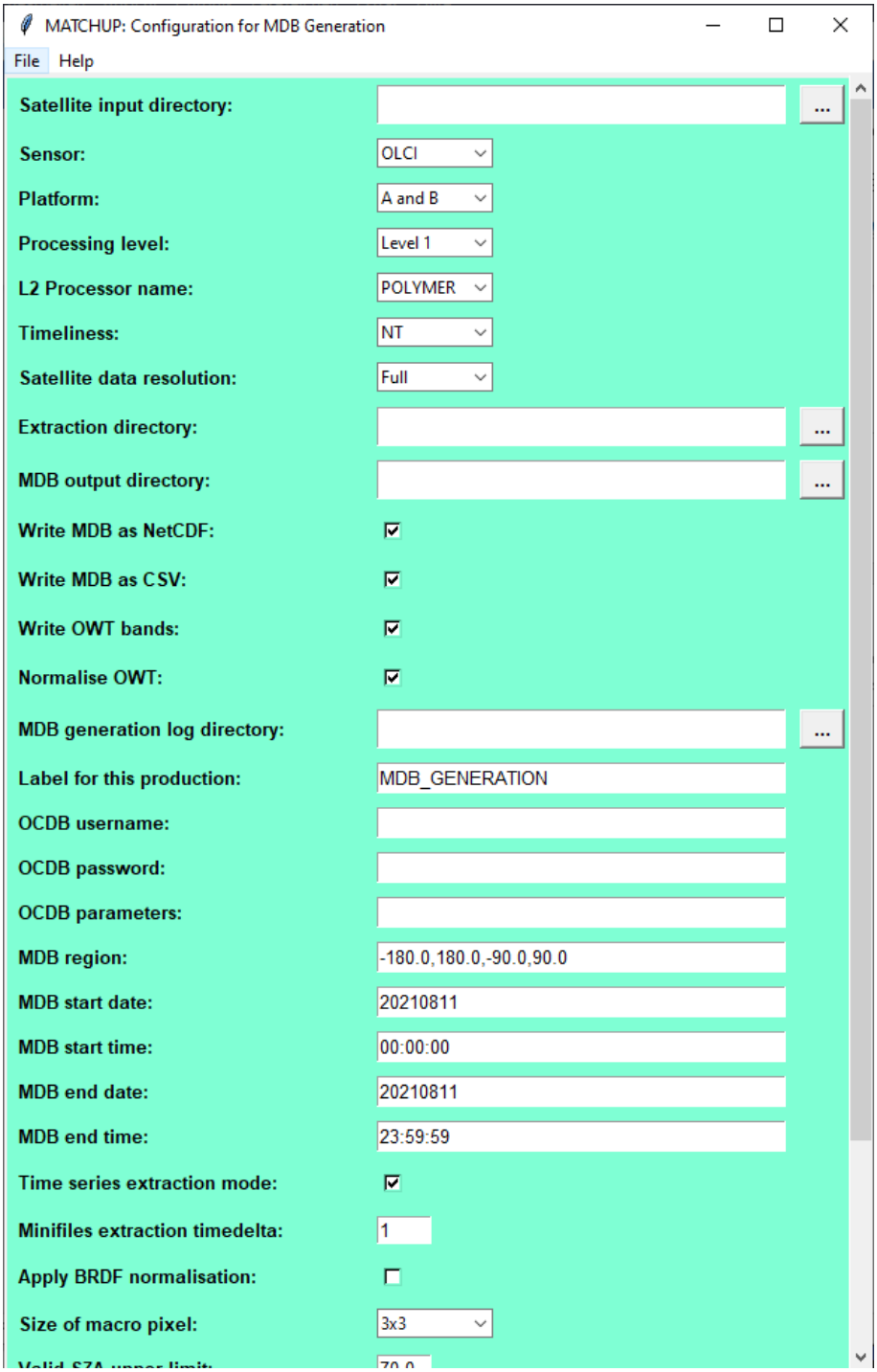

Figure 7: OMAPS configuration tool: Configuration for MATCHUP module

4.6.4.2 Matchup generation

After generation of the config file, the MDB generation can be started from a bash script:

```
# From $OMAPS_ROOT as specified in section 3.1, go to the
```

```
# Offline processor subdirectory:
cd $OMAPS_ROOT/omaps_root/offline_processor

# Start the MDB generation:
./MATCHUP_generate_mdb.bash --configfile=<path/to/config>
```

A log file for the processing run is written into the log directory specified in the configuration.

4.6.4.3 Matchup results

If the processing is successful, the results are written into the MDB output directory. Depending on the configuration, there will be the 'standard' MDB product in NetCDF format, and/or a 'MDB summary file' in CSV format. The format and content of both products are described in full detail in the IODD [AD-8].

4.6.5 Matchup generation as predefined docker operation

There is an option to run a matchup script automatically within a docker container using:

```
python run_docker_container.py <workdir> -r matchup
```

It can be seen in the file $OMAPS_ROOT/omaps_root/container_commands.py that this will call a matchup config file named 'matchup_config_mdb_generation_TEST.ini' located in the 'workspace' folder named in the call. For the matchup script to work all of the locations in the config file for input and output should be contained in the mounted workdir so that they are visible to the docker.

4.6.6 Example test case using the OCDB and matchup module

The following demonstration uses the in-situ dataset from the along with a granule you can create using the steps described in section 4.4. The OLCI level 1 file that we will begin with is 'S3A_OL_1_EFR____20170310T204920_20170310T205120_20171014T033816_0119_015_171_____MR1_R_NT_002.SEN3'. This example provides a cloud free 5x5 macropixel in the central pacific near Hawaii.

In detail, the workflow is:

1) If the granule needs downloading and processing to level 2 then do so as described above by making a workspace and dispatching the docker processing (we will use polymer in this example).

```
#Download the test file from webhost and put somewhere
wget -nH --cut-dirs=3 -r -R "index.html*" --no-parent -e robots=off
https://rsg.pml.ac.uk/shared_files/OMAPS/test_data/S3A_OL_1_EFR____
20170310T204920_20170310T205120_20171014T033816_0119_015_171_____M
R1_R_NT_002.SEN3
mv ./S3A_OL_1_EFR____20170310T204920_20170310T205120_20171014T03381
6_0119_015_171_____MR1_R_NT_002.SEN3 /tmp/
#Make a docker compatible workspace
python $OMAPS_ROOT/omaps_root/ancillary_tools/workspace_creator.py
-dds /tmp/ -fs S3A_OL_1_EFR____20170310T204920* -o /tmp/
#Run the docker with to process L1->L2 using polymer default config
python $OMAPS_ROOT/omaps_root/run_docker_container.py /tmp/workspa
ce_S3A_OL_1_EFR____20170310T204920_20170310T205120_20171014T033816_
0119_015_171_____MR1_R_NT_002.SEN3/ -r polymer_demo
```

2) After the processing has completed you will need to ensure that the output file is visible in a file structure that matches what is expected by the matchup module. The required structure is <satellite_input_dir>/yyyy/MM/dd. We have created a small script that can create the correct structure in the form of symlinks (so no data is actually copied) if you can provide a) a regex that will match all processed files of interest and b) the output location to build the directory structure. See the example below:

```
#create a location where we will build the directories and symlinks
mkdir /tmp/matchup_input_dir_polymer
#Check/update the output filename has a polymer specific tag
find /tmp/workspace_S3A_OL_1_EFR____20170310T204920_20170310T205120
_20171014T033816_0119_015_171_____MR1_R_NT_002.SEN3/9_chl_blending
/output/ -name '*.nc' -exec bash -c ' mv $0 ${0/L2_masked.nc/L2_pol
y.nc}' {} \;
#run the script to create symlinks from final output files
$OMAPS_ROOT/omaps_root/ancillary_tools/create_chrono_dir_struct_sym
links.sh '/tmp/workspace_S3A_OL_1_EFR____20170310T204920_20170310T20
5120_20171014T033816_0119_015_171_____MR1*/9_chl_blending/output/S
3*.nc' /tmp/matchup_input_dir_polymer
#Make the directories for matchups files and log files
mkdir -p /tmp/matchups_output_dir_polymer/rrs/extractions/ocdb/L2_R
RS_polymer/
mkdir -p /tmp/matchups_output_dir_polymer/rrs/mdb/L2_RRS_polymer
mkdir /tmp/matchups_output_dir_polymer/Logs
```

3) Generate MATCHUP configuration with the Configuration tool:
in $OFFLINE_PROCESSOR_ROOT:

```
cd $OMAPS_ROOT/omaps_root/offline_processor
./start_offl_proc_config.bash
```

(If this fails due to ssh display variable access issues then you can copy the example config and edit with your preferred text editing method).
go to MODULE --> MATCHUP
enter config parameters of your choice for matchup generation and save to an ini file. A working config file example is located at:

```
$OMAPS_ROOT/omaps_root/offline_processor/test/demo_configs/matchup_
config_mdb_generation_demo_rrs.ini
```

The config should look something like this:

```
[MDB_GENERATION]
; configuration for mdb generation
satellite_input_dir = /tmp/matchup_input_dir_polymer
extraction_dir = /tmp/matchups_output_dir_polymer/rrs/extractions
mdb_output_dir = /tmp/matchups_output_dir_polymer/rrs/mdb
ocdb_username = olaf
ocdb_password = olaf
l2_processor = POLYMER
processing_label = L2_RRS_polymer
```

```
matchup_variables =
rrs_412:Rrs_412;rrs_443:Rrs_443;rrs_490:Rrs_490;rrs_510:Rrs_510;rrs
_560.5796:Rrs_560;rrs_620.626:Rrs_620;rrs_665:Rrs_665
mdb_region = -180.0,180.0,-90.0,90.0
mdb_start_date = 20170101
mdb_start_time = 00:00:00
mdb_end_date = 20191231
mdb_end_time = 23:59:59
time_delta = 6
sza_max_valid = 70.0
vza_max_valid = 60.0
valid_pixel_expr =
min_valid_pixels = 40.0
variance_factor = 1.5
coeff_of_variation_thresh = 0.5
macro_pixel_size = 5
sensors = OLCI
platforms = A
processing_level = 2
timeliness = NT
resolution = F
brdf = False
log_dir = /tmp/matchups/Logs/
matchup_write_netcdf = True
matchup_write_csv = True
```

If you need information on the options available for setting such as “l2_processor” in this config file then you can look to $OMAPS_ROOT/omaps_root/offline_processor/OFFL_PROCESSOR/matchup/mdb/mdb_builder/mdb_constants.py

4) Generate matchups using the MATCHUP module and the relevant config file:

```
cd $OMAPS_ROOT/omaps_root/offline_processor/
./MATCHUP_generate_mdb.bash
$OMAPS_ROOT/omaps_root/offline_processor/test/demo_configs/matchup_
config_mdb_generation_demo_rrs.ini
```

the MDB (netcdf) file and a csv file (if specified as output option in the config) will be located in the output directories specified:

```
ls /tmp/matchups_output_dir_polymer/rrs/extractions/ocdb/L2_RRS_pol
ymer
>> ocdb_extraction_L2_RRS_polymer_5faebc9b1d2b6d0001788c73.csv
ls /tmp/matchups_output_dir_polymer/rrs/mdb/L2_RRS_polymer
>> MDB_S3A_OLCI_L2_OCDB_L2_RRS_polymer.nc
```

4.7 SVC

As with the other modules the SVC module can be run in an interactive mode or in a more ‘hands-free’ mode (demo). In both cases, the user needs first a Level-1 MDB, to be accessible in the mounted workspace. In the following examples, we consider the OLCI-A MDB at MOBY already generated by EUMETSAT in the OC-SVC-TOOL study, after screening: file MDB_S3A_OLCI_L1_MOBY_screened.nc (not provided in the Docker).

4.7.1 Running as predefined docker operation (demo)

The SVC gains computation can quickly be tested in the docker container for the POLYMER processor using a predefined operation:

```
cd $OMAPS_ROOT/omaps_root
python3 run_docker_container.py <workdir> -r svc
```

This opens the GUI of the OC-SVC-TOOL. Select the "Individual gain computation" tab (middle tab of the GUI). The various fields are predefined with suitable paths to the workspace and POLYMER processor, as defined in the configuration prepared for this demo:
`$OMAPS_ROOT/omaps_root/offline_processor/OFFL_PROCESSOR/tests/svc/ocsvctool_POLYMER.cfg`
Note that this configuration will only launch the SVC on 3 match-ups.

In the GUI you then need to:

- Check the sensor is set to `OLCI-A`
- Select the Level-1 match-up MDB, typically: `/workspace/MDB_S3A_OLCI_L1_MOBY_screened.nc`.
- Let empty the Level-1 PDUs directory, as POLYMER will run in CSV mode and not PDU mode.
- Select the Level-2 wrapper for POLYMER: `/opt/omaps/offline_processor/OFFL_PROCESSOR/svc/oc-svc-tool/wrappers/POLYMER/POLYMER_wrapper.py`
- Check the Level-2 wrapper options are set to: `--sensor OLCI --polymer_home /opt/omaps/external/polymer-v4.13`
- Select the nominal ADF relevant to OLCI: `/opt/omaps/offline_processor/OFFL_PROCESSOR/svc/oc-svc-tool/wrappers/POLYMER/ADF_SVC_OLCI.nc`
- Select only a subset of bands to be calibrated, to limit the computational time. Typically, click on button "`Deselect all bands`" and only ticks two OLCI bands, like 412.5 and 490 nm.
- All other options can be kept as it.
- Identify the Job name, which by default is associated to the sensor and the actual time; the job name is the name of the output directory, created in the workspace.
- Give a free description, that will be displayed on the plots.
- Click button `Go!`

The SVC is then running over the first three match-ups.

When the gains are computed, the GUI can be launched once again to post-process the gains and produce plots:

- Select the "Gains post-processing" tab (third tab of the GUI).
- Check the sensor correspond to that of the demo, i.e. OLCI-A
- Select the SVC job previously finished (name of the output directory)
- If necessary, adjust the screening options. For the POLYMER demo, set the "Max difference between in-situ and satellite" to -1 in order to not discard too many points.
- If necessary, change the post-processing name (this will be the name of the output directory for post-processing, as a subfolder of the previous SVC directory)
- Click button `Go!`

4.7.2 Running in interactive docker

The interactive mode shall be favoured for selecting other processors or tuning more precisely the numerous options of a SVC run. Indeed, while some options can be changed in the GUI, others are fixed in the SVC configuration file, such as number of match-ups, number of iterations, output folder, etc. We refer to the OC-SVC-TOOL user manual for a detailed list of these parameters (https://www.eumetsat.int/ocean-colour-system-vicarious-calibration-tool). The main point is to start from an existing configuration file, copy it and edit it as required. A default configuration file is available at:
`$OMAPS_ROOT/omaps_root/offline_processor/OFFL_PROCESSOR/svc/oc-svc-tool/ocsvctool.cfg`
and two other examples with more dedicated options are also provided in the container:

- `$OMAPS_ROOT/omaps_root/offline_processor/OFFL_PROCESSOR/tests/svc/ocsvctool_POLYMER.cfg`
- `$OMAPS_ROOT/omaps_root/offline_processor/OFFL_PROCESSOR/tests/svc/ocsvctool_SACSO.cfg`

Regarding the processor, the SVC module needs a dedicated wrapper with fixed inputs and outputs; here again with refer to the documentation of the OC-SVC-TOOL for exhaustive description of the interfaces between the SVC module and the processor.

Once this is ready, the basic sequence to launch SVC in interactive mode is as follows:

```
cd $OMAPS_ROOT/omaps_root/offline_processor/OFFL_PROCESSOR/svc/oc-svc-tool/

#Copy locally a config file from the existing template, for instance:
cp  ../../tests/svc/ocsvctool_POLYMER.cfg /workspace/new_svc.cfg

# Edit this configuration file as required (output directory, number of match-ups, etc.)

#Launch SVC GUI with this config file:
python main.py --config /workspace/new_svc.cfg
```

From this step, you can select in the GUI the various options in a similar way to that described in the previous demo, and launch the SVC.

Note that there is a possibility to quickly relaunch a SVC job without filling again the GUI boxes: at the bottom of the GUI, give to `Job name` the name of an existing job. This will restart the job with all options defined in this previous job, whatever the actual options displayed in the GUI. If some match-ups have already been successfully processed in the previous run, they will be skipped. To relaunch the process on all match-ups, you need to remove the directories `nominal_run` and `svc_run`, but keep the configuration file `svc_job.cfg`.

4.8 Atmospheric RR

We assume that you have successfully created matchup extractions with the matchup module. For each atmospheric correction processor you will have to extract one matchup file based on the same in-situ database. As stated earlier it is important that the matchup generation step prior to the AC_RR must only request the bands you wish to compare in the round robin comparison (as the round robin code processes all

bands in the MDB file) and this band set must be the same for all of the atmospheric processor options you wish to compare.

For ease of demonstration we have made 4 matchup files available for download that provide matchups from the OCDB against >700 co-incident granules processed with POLYMER, SACSO, L2GEN and the IPF processor (with brdf applied during matchup extraction). You can download the files with the following commands:

```
mkdir /tmp/OCDB_matchups_RR_test
cd /tmp/OCDB_matchups_RR_test
wget -np -nH -r -R "index.html*" --cut-dirs 3 -i
https://rsg.pml.ac.uk/shared_files/OMAPS/matchups/file_list.txt
mkdir /tmp/OCDB_matchups_RR_test/AC_RR_output
```

There are some adjustments of the configuration file needed. For full information on all the parameters and their properties, please refer to the IODD.
Please edit the default RR-AC configuration file to your needs in your comparison. An example for four atmospheric corrections and their respective matchup extractions can be found at
`$OMAPS_ROOT\omaps\omaps_root\offline_processor\OFFL_PROCESSOR\rr_ac\rrac_config_IPF_SASCO_POLYMER_L2GEN_CBQ.ini`
The configuration file should read like this:

```
[RR_AC]
; configuration for RoundRobin AC
mdb_input_dir = /tmp/OCDB_matchups_RR_test/
mdb_input_files_ipf = MDB_S3A_OLCI_L2_OCDB_L2_RRS_IPF.csv
mdb_input_files_l2gen = MDB_S3A_OLCI_L2_OCDB_L2_RRS_l2gen.csv
mdb_input_files_polymer = MDB_S3A_OLCI_L2_OCDB_L2_RRS_polymer.csv
mdb_input_files_sacso = MDB_S3A_OLCI_L2_OCDB_L2_RRS_sacso.csv
rrac_output_dir_root = /tmp/OCDB_matchups_RR_test/AC_RR_output
processing_label = RRAC_4ACs_test
flag_type = CBQ
rrac_rho_type = rrs
insitu_rho_type = rrs
satellite_rho_type = rrs
check_homogen = False
write_bootstrap_data = True
n_bootstrap = 100
statistical_parameters = MAD,MD,MAPD,MPD,SAM,CHI2
chi2_insitu_bands = rrs_412,rrs_443,rrs_490,rrs_560.5796,rrs_665
chi2_insitu_band_norm = rrs_560.5796
score_parameters = MAD,MD,MAPD,MPD,SAM,CHI2 read_aggregated_data = False
polymer_bits_exclude =
bitmask.LAND,bitmask.CLOUD_BASE,bitmask.L1_INVALID,bitmask.NEGATIVE_BB,b
itmask.OUT_OF_BOUNDS,bitmask.EXCEPTION,bitmask.THICK_AEROSOL,bitmask.HIG
H_AIR_MASS,bitmask.EXTERNAL_MASK,bitmask.CASE2,bitmask.INCONSISTENCY
polymer_bits_include =
sacso_bits_exclude = flags.BAD
sacso_bits_include =
ipf_bits_exclude =
WQSF.CLOUD,WQSF.CLOUD_AMBIGUOUS,WQSF.CLOUD_MARGIN,WQSF.INVALID,WQSF.COSM
ETIC,WQSF.SATURATED,WQSF.SUSPECT,WQSF.HISOLZEN,WQSF.HIGHGLINT,WQSF.SNOW_
ICE,WQSF.AC_FAIL,WQSF.WHITECAPS,WQSF.ADJAC,WQSF.OC4ME_FAIL,WQSF.RWNEG_O2
```

```
,WQSF.RWNEG_O3,WQSF.RWNEG_O4,WQSF.RWNEG_O5,WQSF.RWNEG_O6,WQSF.RWNEG_O7,W
QSF.RWNEG_O8
ipf_bits_include = WQSF.WATER
l2gen_bits_exclude =
l2_flags.ATMFAIL,l2_flags.LAND,l2_flags.CLDICE,l2_flags.SEAICE
l2gen_bits_include =
```

The RR-AC processor expects to find these extraction files in the same folder, which you have to specify in the configuration file (`mdb_input_dir`). All their filenames and the names of the AC processors have to be listed in the same order: here `mdb_input_files = see comma separated list above,` and the four AC processors are listed `mdb_ac_keys = polymer,ipf,sacso,l2gen`. The AC names are used again in finding the correct valid pixel expression, so they have to be consistent with the names given in the valid pixel expression definitions e.g. `validPixelExpression.polymer.exclude.` The valid pixel expressions, which are implemented here, are the default definitions, which have already been applied in the matchup generation and the (independent) statistics, which are calculated during the matchup process. You can change evaluation of the quality flags by removing or adding the AC specific flag names. All expression are combined by AND-statements: if any of the flags in the `exclude` list is raised, the pixel becomes invalid (combination of AND NOT). At the same time, the flag from the `include` list has to be raised, so that the pixel is valid (combination of AND). (Note: If the user wants to apply more complex valid pixel expressions, there is an option for specific definitions envisioned, which would need direct coding.)
Flags can also be applied across all atmospheric corrections in the comparison, so that macropixel values are based on the same pixels for all ACs. To choose this combined valid pixel approach, set `flag_type` to `CBQ` (common best quality; in contrast to `IBQ`, individual best quality).
During the aggregation of the macropixels, which have been extracted by the matchup module, the valid pixels can also be tested on spatial homogeneity (`checkHomogen`).
The user has to choose and set an output path (`rrac_output_dir_root`), where a folder with the specified name (`processing_label`) will be created and all results are stored there: the figures in `$rrac_output_dir_root/processing_label/figures/`, the aggregated data and statistics calculations in `$rrac_output_dir_root/processing_label/results/`, and the automatically generated documentation in `$rrac_output_dir_root/processing_label/doc/`.
Specify, which kind of reflectance type the RR-AC should be based on (`rrac_rho_type` either `rrs` or `rhow`); declare, which reflectance type the insitu data and the AC processed satellite data are respectively (`insitu_rho_type, sat_rho_type`). Conversion will be applied automatically to match the `rrac_rho_type.`
There is a long list of statistical measures available, which can be calculated for each AC macropixel dataset in comparison to the insitu data. Statistical parameters which are taken into account in the inter-comparison are listed in `score_parameters`. If parameters in this evaluation list are not part of `statistical_parameters,` they are automatically added and calculated (`score_parameters` is always a subset of `statistical_parameters`). If the chi-square value between the in-situ and satellite derived spectra is calculated, the names of the spectral bands of the in-situ data have to be listed (`chi2_insitu_bands`, singling out one band for the normalization of the spectra `chi2_band_for_normalisation`, which again takes the name of one of the in-situ bands listed in `chi2_insitu_bands`).
The calculation of statistics and scores is repeated for `N_bootstrap` times, and the results are stored, if `write_bootstrap_data` is true. This allows several diagnostic plots to be created, especially the distributions of statistical parameters.

For large extraction files, the aggregation process of macropixels to a single value can be time consuming. Sometimes, it might be easier to repeat a comparison exercise with already aggregated data (`read_aggregated_data=True`). The aggregated data is supposed to be found in one folder `mdb_input_dir_aggrData`, and the files have to be listed (`mdb_input_files_aggrData`) in the same order as the AC_keys and the original `mdb_input_files`. This is currently necessary, because some information from the matchup processor is only available in the original extraction files, but not in their aggregated counterparts.
The RR-AC code can be started outside a docker by this command (update config file location as required):

```
python $OMAPS_ROOT/omaps_root/offline_processor/omaps_RRAC_main.py --
configfile=$OMAPS_ROOT/omaps_root/offline_processor/OFFL_PROCESSOR/rr_ac
/rrac_config_IPF_SACSO_POLYMER_L2GEN_CBQ.ini
```

4.9 In-water RR

The in-water round robin requires the matching of the in-situ variable measurements to the calculated estimates from the candidate algorithms. An example of a config for such a set of matchups can be found at $OMAPS_ROOT/omaps_root/offline_processor/OFFL_PROCESSOR/tests/matchup/matchup_config_mdb_generation_IW_RR_demo_CHLA.ini. This was used to create a matchup file from hundreds of granules processed using polymer atmospheric correction and all available chlorophyll-a algorithms.

Rather than running the matchup yourself (which would require the processing of all diagnostic granules) have provided the output of the matchup script using the above config. Therefore, the following demonstration of the in-water round robin (IWRR) module begins by downloading this example matchup output.

```
mkdir -p /tmp/OMAPS_IW_RR/matchups_IW_RR_polymer
cd /tmp/OMAPS_IW_RR/matchups_IW_RR_polymer
wget -i https://rsg.pml.ac.uk/shared_files/OMAPS/matchups/file_list2.txt
```

The IWRR config file is described in the IODD and an example is shown below:

```
[Bands]
Band1=412
Band2=443
Band3=490
Band4=510
Band5=560
Band6=665
all_bands=412,443,490,510,560,665
[matchup_files]
match_file_chl=/tmp/OMAPS_IW_RR/matchups_IW_RR_polymer/MDB_S3A_OLCI_L2_O
CDB_IW_RR_L2_FULLTEST_CHLA.csv
match_file_iop=NaN
match_file_kd=NaN
match_file_rrs=/tmp/OMAPS_IW_RR/matchups_IW_RR_polymer/MDB_S3A_OLCI_L2_O
CDB_IW_RR_L2_FULLTEST_CHLA.csv
[candidate_algorithms]
chl_algorithms:
chlor_oc2,chlor_oc3,chlor_oc4,chlor_ocx,chlor_oci,chlor_oci2,chlor_oc5,c
hlor_oc5ci
[RR_options]
```

```
RR_type: chl
insitu_column_string: chla_fluor
bootstraps: 100
LogNormVar: True
SplitByWaterClass: False
WeightedByWaterClass: False
threshold_memb: 0.3
Split_Criteria: dominant
[water_class_options]
omaps_root = /users/thomasjackson/PROJECTS/OMAPS/Git_Code/
owt_set_file =
$OMAPS_ROOT/omaps_root/offline_processor/OFFL_PROCESSOR/rr_w/algorithm_d
ependencies/modules/OWT_classification/example_files/example_6band_norma
lised_clusters.nc
owt_normalisation = True
[output_options]
Plots: True
Tables: True
Directory: /tmp/OMAPS_IW_RR/output/
```

You can then simply run the IWRR using a config file and specifying the input data type (csv, MDBcsv) with:

```
conda activate omaps
python -W ignore
$OMAPS_ROOT/omaps_root/offline_processor/OFFL_PROCESSOR/rr_w/Round_Robin
_in_water.py -c
$OMAPS_ROOT/omaps_root/offline_processor/OFFL_PROCESSOR/rr_w/example_RR_
config_docker -ft MDBcsv
```

The configuration file used for the extraction of the example IWRR test dataset did not contain a request for ‘oci’ algorithm estimates. This was done on purpose so that when combined with the demonstration IWRR config above it should demonstrate the addition of the oci data to the output file.

Running this demonstration script should provide outputs in the directory /tmp/OMAPS_IW_RR/output. Due to the random nature of the bootstrapping the precise length of the error bars etc on the performance summary plot may differ, but otherwise you should see plots similar to those shown below.

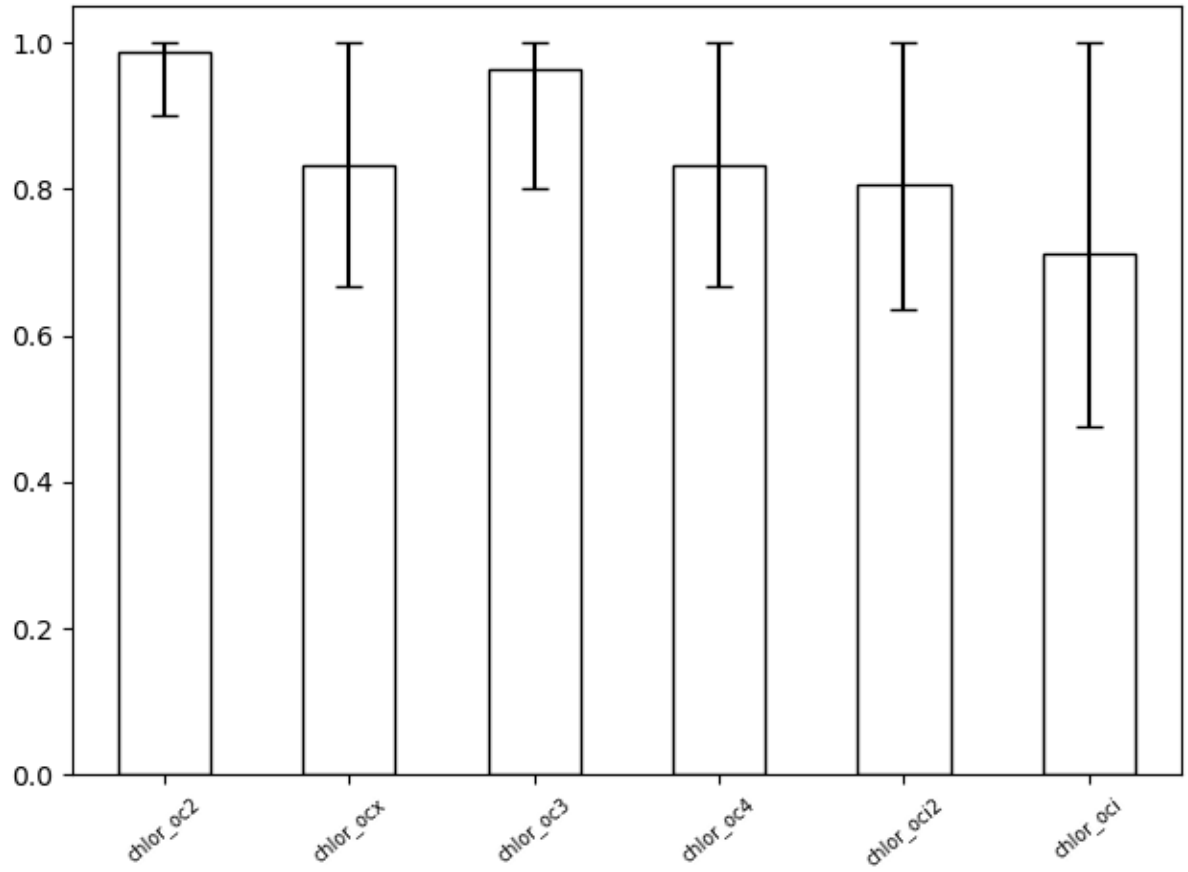

Figure 8: Example summary plot of multi-metric perforamance scores across a range of algorithms with bootstrapping.

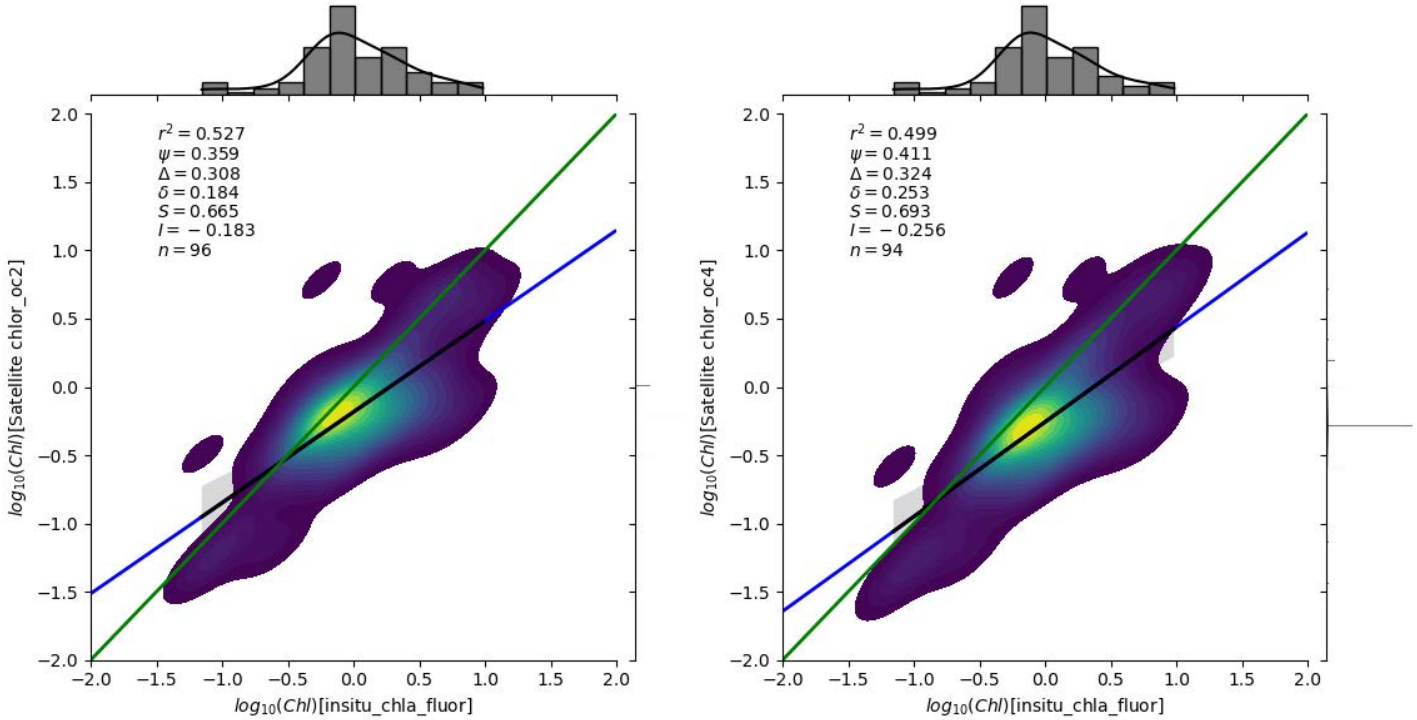

Figure 9: Example plots of single algorithm performance for all matchups available.

There is also a second demo configuration file that is set up to perform an IWRR analysis per waterclass ($OMAPS_ROOT/omaps_root/offline_processor/OFFL_PROCESSOR/rr_w/example_RR_config_docker2) but the example matchups generated do not contain enough matchups to give a full assessment for all waterclasses. The code will still function but will only generate summary plots for waterclasses with >20 matchups.

4.10 Product validation

4.10.1 Matchup statistics

Following the OMAPS requirements, matchup statistics (numbers and plots) for any of the matchup variables shall be provided. A set of statistical quantities is already provided with the CSV summary file generated by the MATCHUP module (see section 4.6.4.3 and IODD), further numbers are given within the scatter plots provided by the VAL module of the Offline processor, as described below.

4.10.1.1 Scatter plot configuration

The VAL module of the Offline processor provides a plot utility for the generation of sophisticated scatter plots of matching in situ / satellite variables. A variety of properties of these plots can again be defined by the user via a configuration ini file. Again, it is recommended to generate the file with the OMAPS configuration tool.

The tool is started as described earlier for the other modules:

```
# From $OMAPS_ROOT as specified in section 3.1, go to the
# Offline processor subdirectory:
cd $OMAPS_ROOT/omaps_root/offline_processor

# Start the GUI:
./start_offl_proc_config.bash
```

The main screen of the configuration tool should appear. From the menu, the entries ‘Module → VAL → Configure MDB scatterplot generation’ must now be selected, then the configuration table should appear. Again, the necessary entries can be made here and can be saved (and reloaded if needed) via the ‘File’ menu (Figure 6).

As we can see from Figure 10, there are again lots of configuration parameters. A complete table with possible values and descriptions of all these parameters is given in the OMAPS IODD [AD-8], together with an example of a configuration ini file. In the configuration GUI (Figure 6), a short description for each parameter is provided with tooltips. However, most of them are self-explaining.

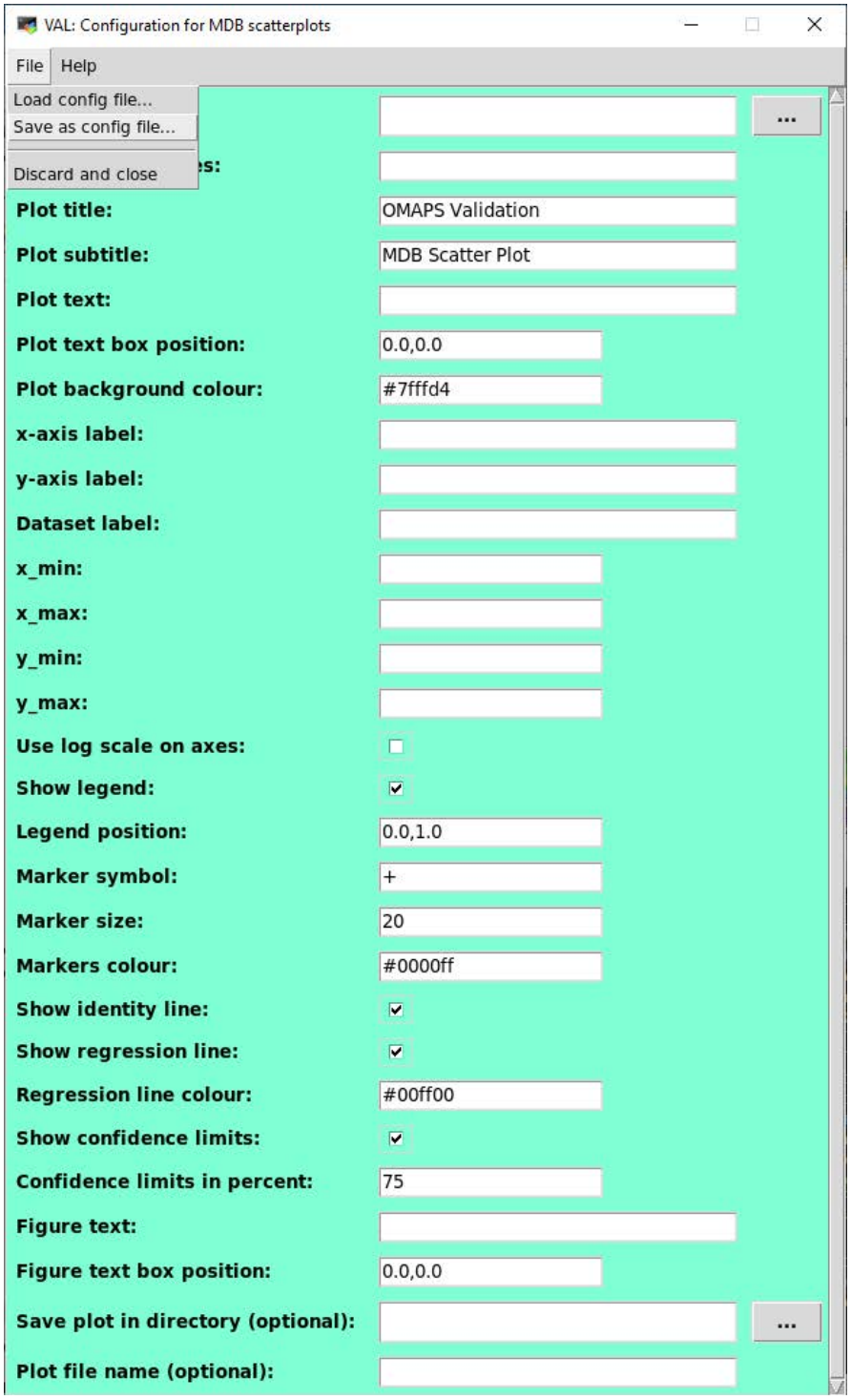

Figure 10: OMAPS configuration tool: Configuration for scatter plots of matchup variables

4.10.1.2 Scatter plot generation

After generation of the config file, the generation of the scatter plot can again be started from a bash script:

```
# From $OMAPS_ROOT as specified in section 3.1, go to the
# Offline processor subdirectory:
```

```
cd $OMAPS_ROOT/omaps_root/offline_processor

# Start the scatter plot generation:
./VAL_scatterplot_mdb.bash --configfile=<path/to/config>
```

4.10.1.3 Examples

Figure *11* shows an example of a scatter plot which was generated as described above. The plot shows OLCI L2 (IPF) vs. OCDB in situ for Rrs(443nm). The region is around Hawaii (MOBY). The time interval is 20170201-20180430, but only a subset of 10 OLCI L2 products was used here as input. Plot features include a description text field, a regression line, identity line and various statistical parameters. The mosaic Figure 12 in shows a comparison of IPF, SACSO and POLYMER processors vs. OCDB in situ for Rrs(443nm) and Rrs(443nm). Each of the six single plots was generated as the one in Figure 11.

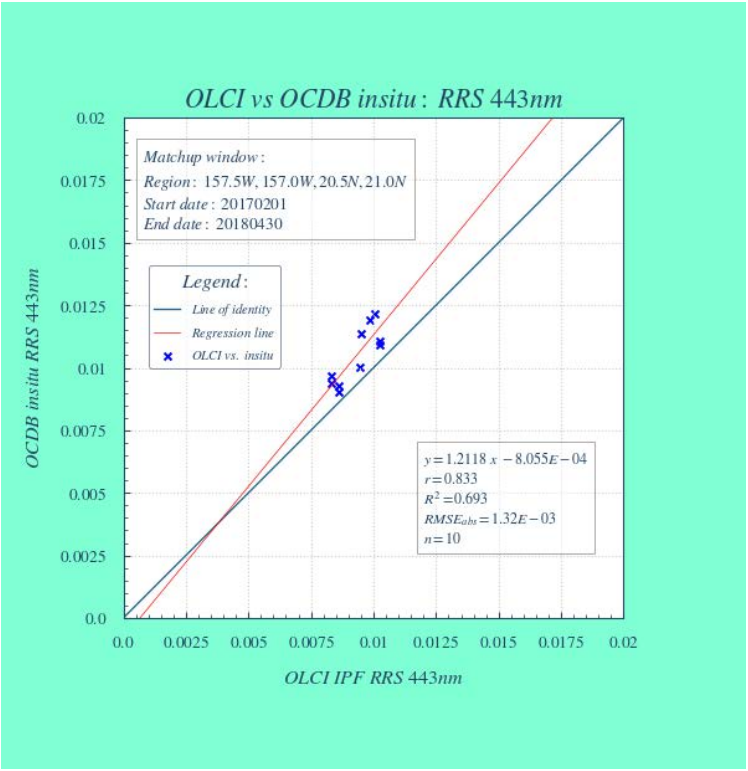

Figure 11: Scatterplot of OLCI L2 (IPF) vs. OCDB in situ for Rrs(443nm). The region is around Hawaii (MOBY). Time interval is 20170201-20180430, but only a subset of 10 OLCI L2 products was used here as input.

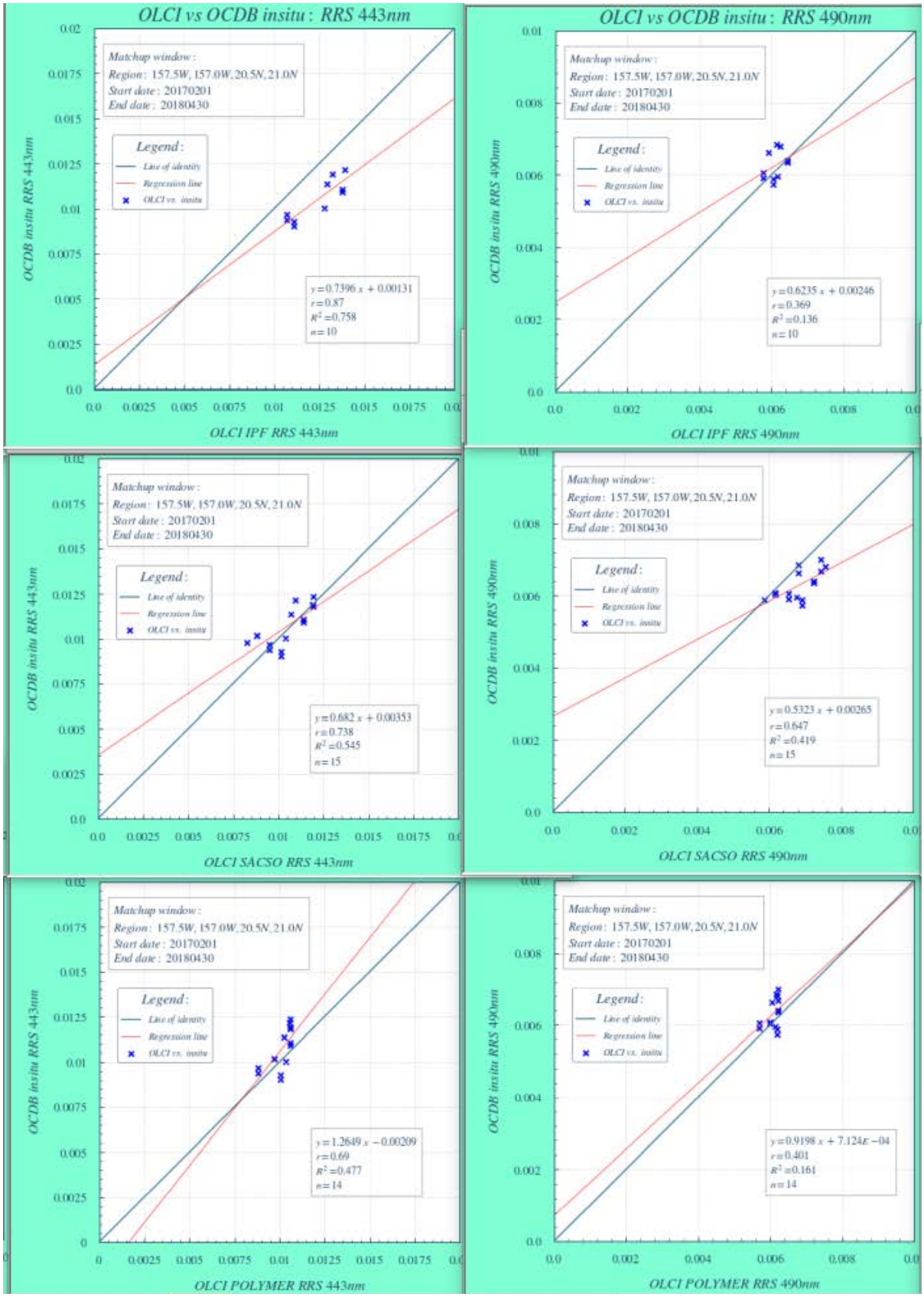

Figure 12: Scatterplots of Rrs(443nm) and Rrs(490nm) for three L2 processors (IPF, SACSO, POLYMER) vs. OCDB in situ. Same region, time interval and subset of OLCI products as in Figure 11 was used as input.

4.10.2 PVER plots

There are a number of figures contained within the OMAPS PVER document that were created using the data from the OMAPS processor to show product performance and validation statistics of different processing configurations. Such plots can be recreated using scripts within the online_processor, offline_processor and ancillary_tools directories and we will demonstrate examples of this below.

The first example is how to create a 'comparative diagnostic granule plots' as shown in Figure 13.

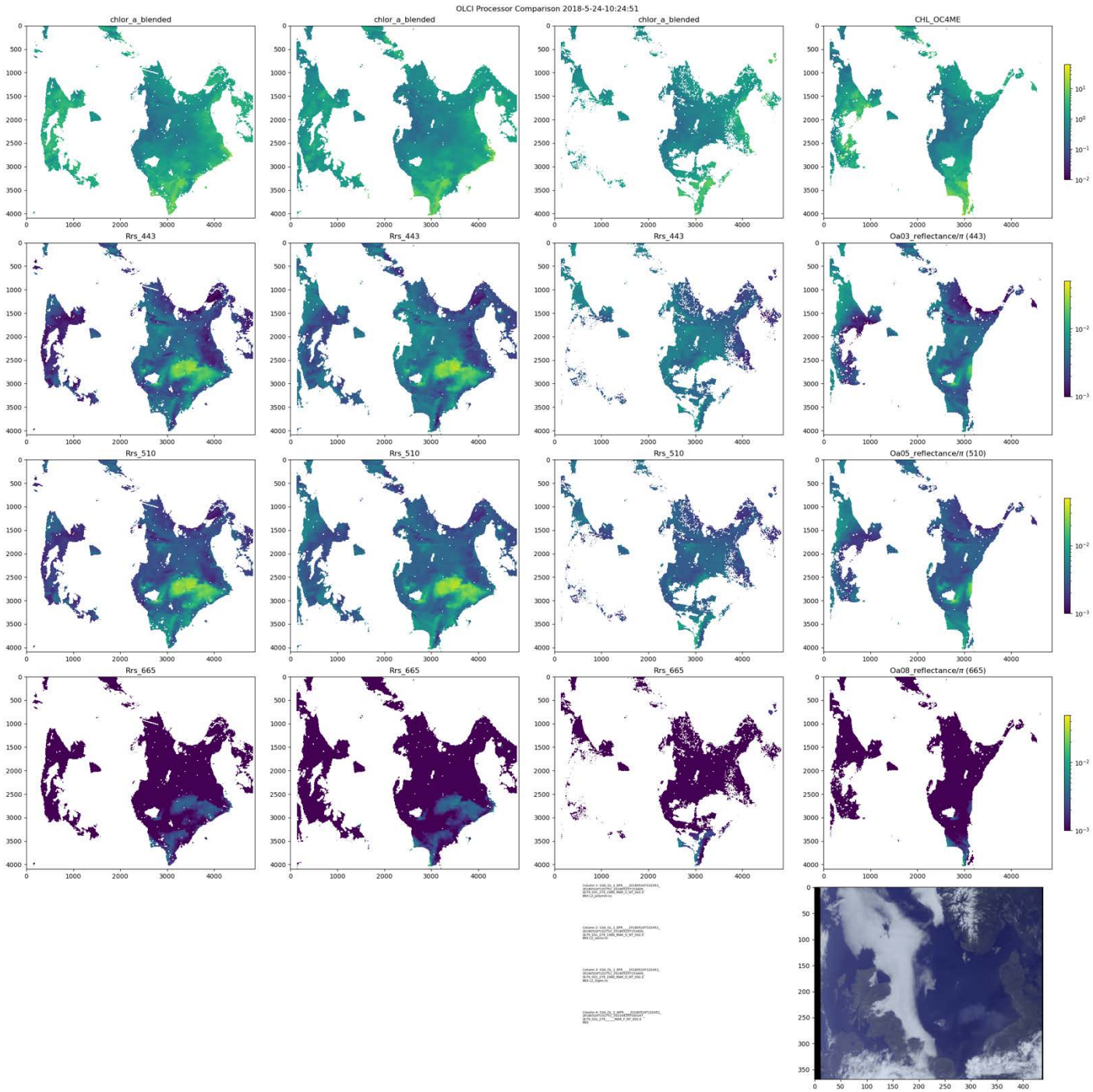

Figure 13: Comparison of diagnostic granule 'S3A_OL_1_EFR____20180524T102451_20180524T102751_*' when processed using (left to right) POLYMER, SACSO, L2GEN and BASELINE processing schemes. Lower right shows a Tristimulus image generated from the level 1 TOA radiance data.

Begin by processing the candidate level 1 granule using each of the 4 processing candidates. This should be done either through creating your own configuration files or using the default sacso, polymer and l2gen demo configurations as covered in section 4.4. You will also need to download (or have access to) the latest EUMESAT baseline L2 processing of the same granule.

Next we will run the comparative plotter script assuming (for simple demonstration purposes) that you have the differing processing outputs located at:

1. /data/output/polymer/workspace_S3A_OL_1_EFR____20180524T102451_20180524T102751_20180525T153409_0179_031_279_1980_MAR_O_NT_002.SEN3/9_chl_blending/output/S3A_OL_1_EFR____20180524T102451_20180524T102751_20180525T153409_0179_031_279_1980_MAR_O_NT_002.SEN3.L2_polymer.nc
2. /data/output/sacso/workspace_S3A_OL_1_EFR____20180524T102451_20180524T102751_20180525T153409_0179_031_279_1980_MAR_O_NT_002.SEN3/9_chl_blending/output/S3A_OL_1_EFR____20180524T102451_20180524T102751_20180525T153409_0179_031_279_1980_MAR_O_NT_002.SEN3.L2_sacso.nc
3. /data/output/l2gen/workspace_S3A_OL_1_EFR____20180524T102451_20180524T102751_20180525T153409_0179_031_279_1980_MAR_O_NT_002.SEN3/9_chl_blending/output/S3A_OL_1_EFR____20180524T102451_20180524T102751_20180525T153409_0179_031_279_1980_MAR_O_NT_002.SEN3.L2_l2gen.nc
4. /data/output/EUMETSAT_baseline_3/S3A_OL_2_WFR____20180524T102451_20180524T102751_20210429T030147_0179_031_279_____MAR_F_NT_003.SEN3

And the corresponding level1 input file at:

/data/input/OLCI_S3A/2018/05/24/S3A_OL_1_EFR____20180524T102451_20180524T102751_20180525T153409_0179_031_279_1980_MAR_O_NT_002.SEN3

You would then call the plotting script with:

```
conda activate omaps
cd $OMAPS_ROOT/omaps_root
python ./ancillary_tools/OLCI_Granule_comparison_plotter.py -f1
‘/data/output/*/workspace_S3A_OL_1_EFR____20180524T102451*/9_*/output/S3A*.nc’ -f1 ‘/data/output/EUM*/ S3A_OL_2_WFR____20180524T102451*’ -l1 ‘/data/input/OLCI_S3A/2018/05/24/ S3A_OL_1_EFR____20180524T102451*’ -o /tmp/ -kf 'WATER,INLAND_WATER' -rf 'CLOUD,CLOUD_AMBIGUOUS,CLOUD_MARGIN,INVALID,COSMETIC,SATURATED,SUSPECT,HISOLZEN,HIGHGLINT,SNOW_ICE,AC_FAIL,WHITECAPS,ADJAC,RWNEG_O2,RWNEG_O3,RWNEG_O4,RWNEG_O5,RWNEG_O6,RWNEG_O7,RWNEG_O8,OC4ME_FAIL' --logvars CHL_OC4ME
```

where:

- multiple -f1 arguments can be used to add regex expressions that match files you wish to compare (this can be any number of files, each will just add a column to the output plot (but the final files must have unique filenames, note the _<processor>.nc at the end of each of files 1-3 above).
- The -l1 argument is a regex for the corresponding level 1 file.
- The -o argument is the output directory
- The -kf argument is EUMESAT BASELINE flags that are required for data to be valid (water in this case)
- The -rf argument is EUMETSAT BASELINE flags that are mean data should be masked.
- The --logvars argument specifies variables that are stored as a log variable and will require unlogging before plotting (such as EUMETSAT BASELINE CHL_OC4ME data).

This script can easily be wrapped in a bash wrapper to process a list of granules stored in a text file for batch plotting. An example might be:

```
#!/bin/bash
Granule_list_file=$1

for file in $(awk -F "," 'NR>0 { print $1 }' $Granule_list_file | awk '{gsub(/ /,""); print}')
do
   echo 'looking for file using name ' $file
   file_regex_set_1="'/data/output/*/workspace_${file}}*/9_*/output/S3A*.nc'"
   file_regex_eum="'/data/output/EUMETSAT_baseline_3/S3*${file:12:36}*'"
   l1file="'/data/input/OLCI_S3A/${file:16:4}/${file:20:2}/${file:22:2}/*${file:12:36}*'"
   echo 'level 1 file : ' $l1file
   python /users/rsg/thja/PROJECTS/OMAPS/OMAPS_testing/OLCI_Granule_comparison_plotter.py -f1 $file_regex_set_1 -f1 $file_regex_eum -l1 $l1file -o /tmp/ -kf 'WATER,INLAND_WATER' -rf 'CLOUD,
CLOUD_AMBIGUOUS,CLOUD_MARGIN,INVALID,COSMETIC,SATURATED,SUSPECT,HISOLZEN,HIGHGLINT,SNOW_ICE,AC_FAIL,WHITECAPS,ANNOT_ABSO_D,ADJAC,RWNEG_02,RWNEG_03,RWNEG_04,RWNEG_05,RWNEG_06,RWNEG_07,
RWNEG_08,OC4ME_FAIL' --logvars CHL_OC4M

done
```

Where the Granule_list_file contains unique filename strings such as
S3A_OL_1_EFR____20180524T102451_20180524T102751_
S3A_OL_1_EFR____20180716T073803_20180716T074103_
S3A_OL_1_EFR____20180730T080327_20180730T080627_

In the second example we will create a time series plot over a fixed location for comparison to an in-situ time series dataset.
For example, after processing all granules that overlap with the Hawaiian Ocean Time Series (HOTS) station ALOHA a timeseries was plotted comparing the in-situ station data against the satellite time series. This process takes 5 steps.

1) Generate a list of files that overlap with the location of interest

```
python3 run_docker_container.py $(pwd)
python ./offline_processor/OLCI_granule_finder.py --configfile
./offline_processor/test/demo_configs/OLCI_granule_finder_test_ALOHA.i
ni > /tmp/ALOHA_granules.txt
```

2) Process the list of granules with the processor configuration of interest (e.g POLYMER demo config used here) with masking applied. It is assumed that the input Level1 granules are in /data/input/S3* . WARNING THE BELOW COMMAND COULD SPAWN A LOT OF DOCKER PROCESSING.

```
while IFS= read -r file_string; do
python $OMAPS_ROOT/omaps_root/ancillary_tools/workspace_creator.py -
dds /data/input/ -fs ${file_string:0:32} -o /data/output/polymer/

workdir=/data/output/polymer/workspace_${file_string:0:32}*

python run_docker_container.py <workdir> -r polymer_demo
done < /tmp/ALOHA_granules.txt
```

3) Once complete create a list of files for extraction.

```
while IFS= read -r file_string
do realpath
/data/output/polymer/workspace_${file_string:0:32}*/9_*/output/*.nc >>
/tmp/process_ALOHA_granules_polymer.txt
done  < /tmp/ALOHA_granules.txt
```

Note the equivalent for the BASELINE files would be:

```
while IFS= read -r file_string
do realpath
/data/output/EUMETSAT_baseline_3/${file_string:0:4}*${file_string:12:2
0}* >> /tmp/processed_ALOHA_granules_baseline.txt
```

```
done  < /tmp/ALOHA_granules.txt
```

4) Run the extractions on these also

```
python ./ancillary_tools/point_extraction_for_timeseries.py -i
/tmp/processed_ALOHA_granules_polymer.txt -la 20.816667 -lo -
157.191667 -o /tmp/extracted_moby_polymer.csv -t S3_MERGED -v
'chlor_a_blended,Rrs_443,Rrs_510,Rrs_665' -N 5
```

Note the equivalent for the BASELINE files would be:

```
python ./ancillary_tools/ point_extraction_for_timeseries.py -i
/tmp/processed_ALOHA_granules_baseline.txt -la 20.816667 -lo -
157.191667 -o /tmp/extracted_moby_baseline.csv -t  S3_FOLDER -v
'chl_oc4me,Oa03_reflectance,Oa05_reflectance,Oa08_reflectance' -N 5 -m
'CLOUD,CLOUD_AMBIGUOUS,CLOUD_MARGIN,INVALID,COSMETIC,SATURATED,SUSPECT
,HISOLZEN,HIGHGLINT,SNOW_ICE,AC_FAIL,WHITECAPS,ANNOT_ABSO_D,ANNOT_MIXR
1,ANNOT_DROUT,ANNOT_TAU06,RWNEG_O2,RWNEG_O3,RWNEG_O4,RWNEG_O5,RWNEG_O6
,RWNEG_O7,RWNEG_O8,OC4ME_FAIL' -km 'WATER,INLAND_WATER'
```

5) Plot the time series of various sets of satellite data variables against the relevant in-situ variables:

```
python ./ancillary_tools/PVER_time_series_plotter_example.py -i
/tmp/extracted_moby_l2gen.csv /tmp/extracted_moby_polymer.csv
/tmp/extracted_moby_sacso.csv -r
/tmp/test_insitu/insitudb_v7_rrs_satbands2.txt -c 'rrs_dataset:moby' -
o /tmp/PVER_TS_PLOTS -v 'Rrs_510' -iv 'rrs_OLC4' -l 'MOBY_Rrs_510'

python ./ancillary_tools/PVER_time_series_plotter_example.py -i
/tmp/extracted_moby_baseline.csv -r
/tmp/test_insitu/insitudb_v7_rrs_satbands2.txt -c 'rrs_dataset:moby' -
o /tmp/PVER_TS_PLOTS -v 'Oa05_reflectance' -iv 'rrs_OLC4' -l
'MOBY_Rrs_510'

python ./ancillary_tools/PVER_time_series_plotter_example.py -i
/tmp/extracted_moby_baseline.csv -r
/tmp/test_insitu/insitudb_v7_chl.csv -c 'chla_fluor_dataset:hot' -o
/tmp/PVER_TS_PLOTS -v 'chl_oc4me' -iv 'chla_hplc,chla_fluor' -l
'HOTS_CHLA_Fluor'

python ./ancillary_tools/PVER_time_series_plotter_example.py -i
/tmp/extracted_moby_l2gen.csv /tmp/extracted_moby_polymer.csv
/tmp/extracted_moby_sacso.csv -r /tmp/test_insitu/insitudb_v7_chl.csv
-c 'chla_fluor_dataset:hot' -o /tmp/PVER_TS_PLOTS -v 'chlor_a_blended'
-iv 'chla_hplc,chla_fluor' -l 'HOTS_CHLA_Fluor'
```

This would then generate images such as those shown below, noting that the date ranges of the granules will be set by the data availability in coda and codarep:

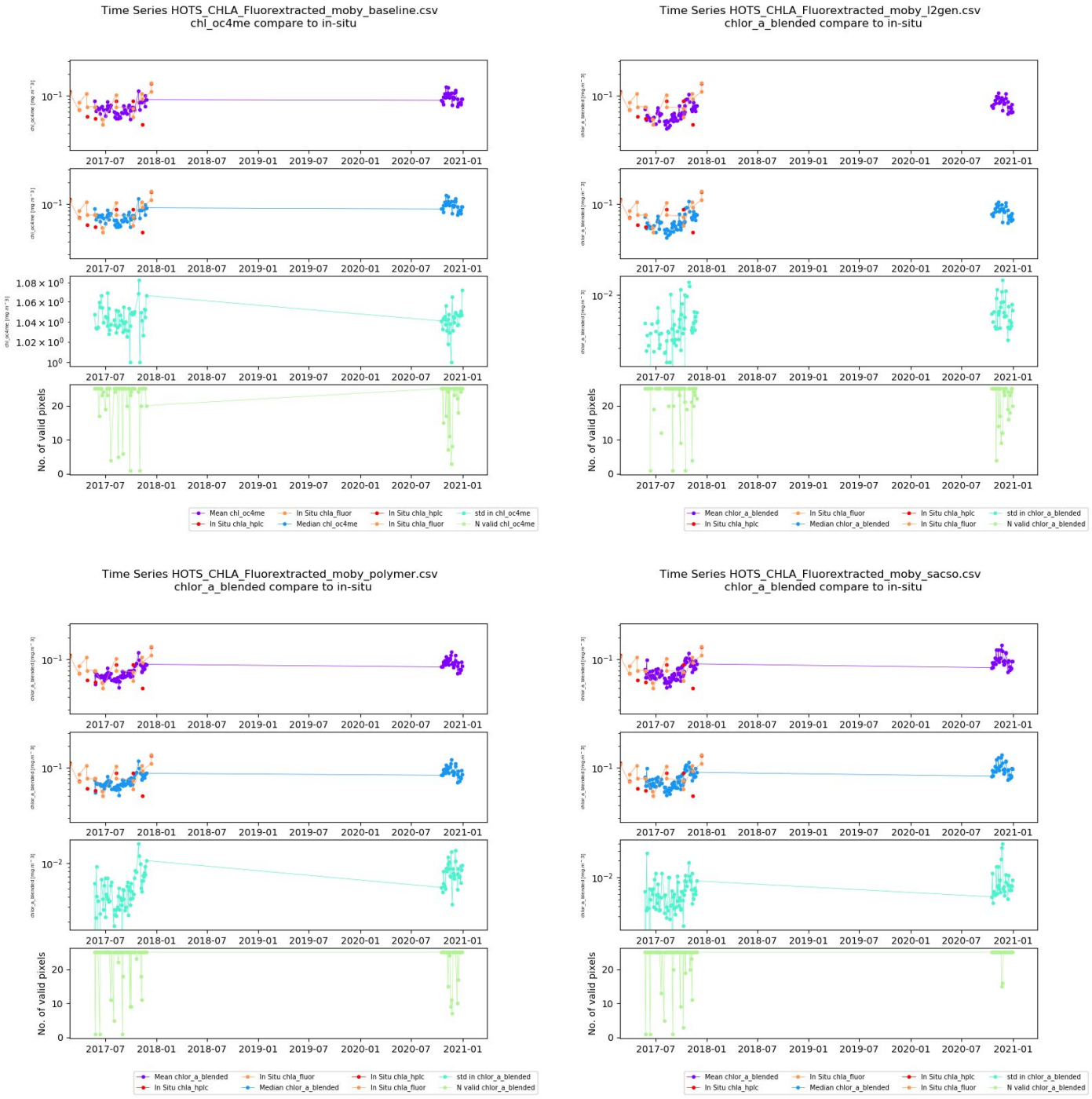

Figure 14:Comparisons of differing Rrs time-series from multiple processor configurations with data extracted from a fixed sample (HOTS) location, compared to the in-situ time series.

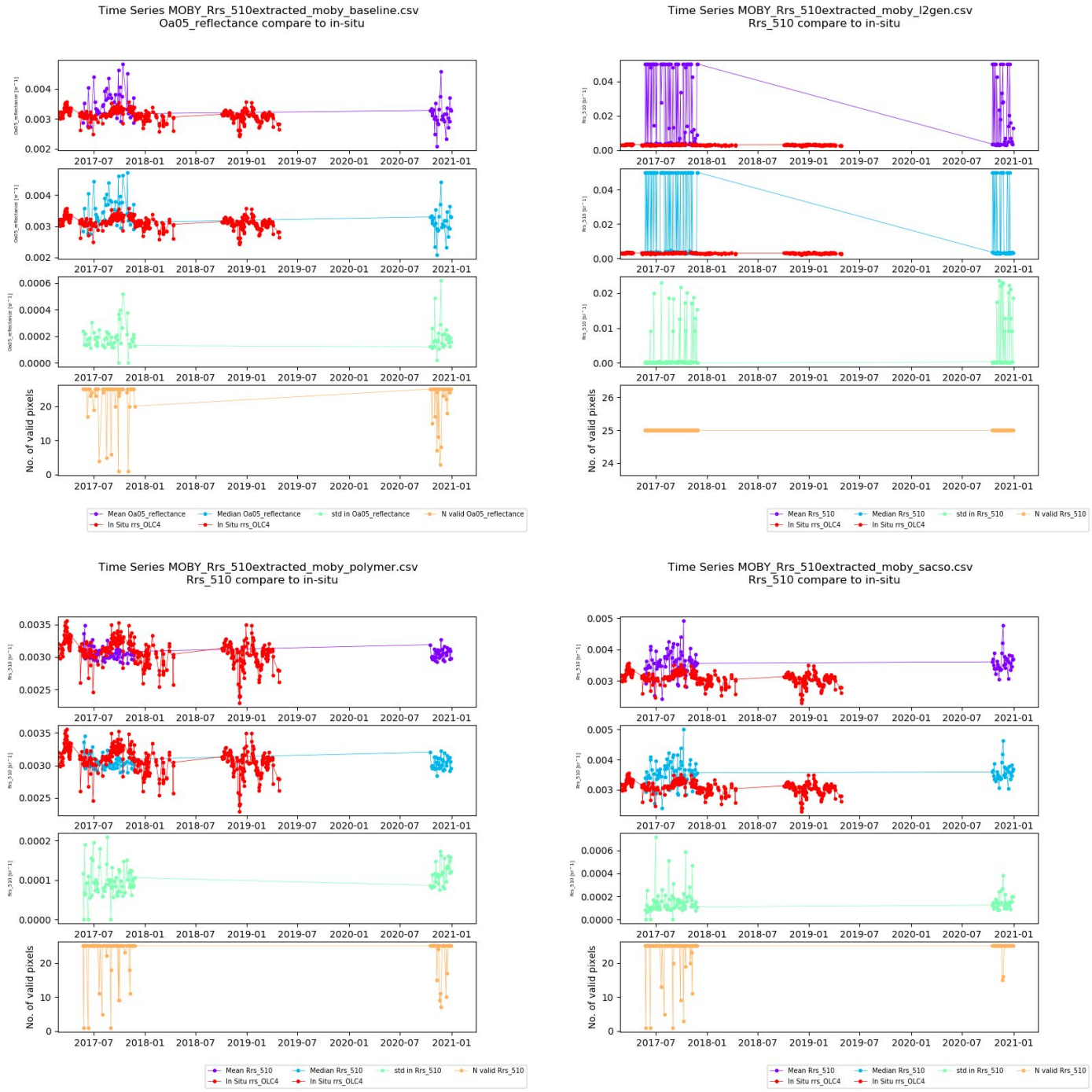

Figure 15: Comparisons of differing Rrs time-series from multiple processor configurations with data extracted from a fixed mooring (MOBY) location, compared to the mooring time series.

5 Summary

5.1 Further development and help for users

This software package is designed to allow the flexible processing and comparison of OLCI products. It is intended that the software will be updated in the future to incorporate additional processing options as new algorithms become available. We also hope to include enhanced modularity within the online processor.

Processing component	Lead Developer
Online processor	James Dingle
Offline Processor	Olaf Danne
Docker environment	Ben Calton
Polymer and Sacso processors	Francois Steinmetz

The dev team have bi-weekly coding meetings and if you require an update or assistance with any of the code listed above then please contact the team via the #support channel in the OMAPS Slack group at https://omaps.slack.com

6 FAQ

Q: My docker build fails saying “ubuntu archive is unsigned”?
A: This "ubuntu archive" is unsigned error is usually related to not having enough space on your disk.

If you are still running into issues of space on the virtual device due to multiple builds etc then you will need to look up:

```
docker volume ls
docker volume ls -qf dangling=true
or
docker rmi
```

which you can use to remove autogenerated images that are not used.
If when you try to rm an older docker image you get a warning about a closed container using the image you can remove all the containers with:

```
docker rm  $(docker ps -q -a)
```

Additionally, if you have rebuilt many docker images during testing etc then you may need to use:

```
docker image prune
```

to remove dangling images (which can potentially take up 10’s or 100’s of GB of memory).

Q: My docker build fails at stage 18 during snap install?
A: You can run also into an issue with snap installation during the build due to cache history of the docker (if you have built things before). To stop this (usually a fail at stage 18) then you can use --no-cache on the end of the build command.

Q: I am getting an error in which the docker doesn’t have write permissions to the workspace.
A: Check to see if the location that you have mounted as the workspace is local to the host machine or is a network storage location. NFS servers will likely not allow the docker to write to a networked workspace as it will not be owned by the docker ‘user’.

Q: I am getting a disk full error when trying to build a singularity image.
A: This might mean that your local disk is full or that you have a cap on the size of your /tmp/ folder (the default building dir). If your hard drive is full then you will need to free up space. If the issue is the /tmp/ folder limit then you can use the –tmpdir flag to set a new location for intermediate build products. For example :

```
sudo singularity build --tmpdir ~/big_scratch_space/
~/singularity_output_dir/omaps.sif docker-
daemon://omaps/omaps_precessor:latest
```

Q: When is try to start the docker I get the following message:

Server:

ERROR: Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?

A: This means that your docker program is not running (so you cannot run docker images). Run the following in a terminal to initialise docker (on a unix system).

sudo service docker start

or

open /Applications/Docker.app (on Mac OS)

Q: My hard drive is filling up as the output products from the processor are very large.

A: Some of the processing configurations, such as sacso, write a lot of intermediate variables to the file by default. The list of variables can be reduced using the configuration file at XXXXXXXX.