CIMR L2 Sea Ice Drift ATBD v1

Thomas Lavergne and Emily Down, Norwegian Meteorological In

Jun 28, 2024

CONTENTS

This document describes the algorithm theoretical basis for the Sea Ice Drift (SID) Level-2 product.

CHAPTER

ONE

ABSTRACT

Pushed by winds and ocean currents, sea ice is always on the move. Satellite remote sensing is an effective tool to monitor this motion, especially from microwave imagery.

Monitoring of sea ice drift vectors by means of passive microwave is commonly based on motion tracking algorithms working from pairs of brightness temperature imagery in both horizontal and vertical polarization. Depending on the satellite missions, different microwave frequencies have been used, but one generally expects higher accuracy from the higher resolution channels. Accordingly, this version of the Sea Ice Drift Level-2 product focuses on the K and Ka imagery.

Contrarily to other existing operational sea-ice drift product (such as those of IFREMER Cersat or EUMETSAT OSI SAF), the CIMR sea-ice drift product is a Level-2 product. It is thus computed from pairs of overlapping swaths, not from pairs of daily averaged maps. This "swath-to-swath" setup was introduced in Lavergne *et al.* [2021].

Key assets of CIMR in terms of Sea Ice Drift monitoring are 1) its increased spatial resolution at K- and Ka-band (more accurate drift vectors), 2) larger swath (more drift vectors) and 3) the forward and backward scans (better quality control). The main source of uncertainty for the CIMR Sea Ice Drift product will probably be the geolocation uncertainty, hence the importance of the pointing accuracy (in orbit) and geolocation correction steps (in Level-1).

Sea-ice drift vectors are assessed against trajectories from in situ buoy deployed on the ice. A sufficient number of these buoys must be ensured throughout the lifetime of the mission, both in the northern and southern hemisphere. Sea-ice drift vectors from Synthetic Aperture Radar (SAR) as derived in the Copernicus Marine Service Sea Ice Thematic Assembly Center from Sentinel-1 (in the future S1-NG and ROSE-L) can also be used for validation.

CHAPTER

TWO

APPLICABLE AND REFERENCE DOCUMENTS

List of applicable documents (AD) and reference documents (RD).

ID	Document
AD-1	C. J. Donlon, 2019. The Copernicus Imaging Microwave Radiometer (CIMR) Mission Requirements Document v4.0, available here

CHAPTER

THREE

ACRONYMS

Here follows a formatted list of acronyms

ATBD
: Algorithm Theoretical Basis Document

EASE2
: *Equal Area Scalable Earth v2*

IODD
: Input/Output Data Definition

PSD
: Product Specification Document

PVP
: Product Validation Plan

MRC
: Mission Requirement Consolidation

SIC
: *Sea Ice Concentration*

CIMR
: *Copernicus Imaging Microwave Radiometer*

TB
: *Brightness Temperature*

CHAPTER

FOUR

DEFINITIONS USED IN THIS ATBD

Here follows a glossary, or list of definitions

Sea Ice Concentration
The Sea Ice Concentration is the fraction of the sea surface covered by sea ice. It is a measure of the amount of sea ice in a given area. It is usually expressed as a percentage.
Sea Ice Motion
Sea Ice Drift
Sea Ice Motion and Sea Ice Drift are used interchangeably to characterize the horizontal motion of sea ice above the ocean. It is at minimum described by a vector.
Copernicus Imaging Microwave Radiometer
The Copernicus Imaging Microwave Radiometer is a Microwave Radiometer which launch is planned by ESA in 2028.
Brightness Temperature
The Brightness Temperature is the temperature of a surface as seen by a passive microwave sensor. It is a measure of the amount of energy emitted by a surface. It is usually expressed in Kelvin.
Equal Area Scalable Earth v2
Equal area projection grid definition based on a Lambert Azimuthal projection and a WGS84 datum. Defined in [Brodzik *et al.*, 2012].

CHAPTER

FIVE

INTRODUCTION, PURPOSE AND SCOPE

The balance between air drag, ocean drag and lateral forces controls the motion of sea ice [Leppäranta, 2011]. At the local scale, sea-ice motion can both be a facilitator and impediment to ship navigation, opening and closing routes, opening leads, or forming pressure ridges. At the larger regional to basin scales, sea-ice motion (a.k.a. sea-ice drift) exports sea ice to lower latitudes where it melts, contributing to the redistribution of fresh water [Haine, 2015]. Inside the Arctic Ocean, re-circulation of sea-ice (e.g. in the Beaufort Sea) leads to the ageing and thickening of the ice pack towards the northern coasts of the Canadian Arctic Archipelago and Greenland [Timmermans and Marshall, 2020]. Sea-ice drift also plays a role in sea-ice formation and ocean circulation via the formation of coastal latent heat polynyas [Ohshima *et al.*, 2016], as well as in the transport of sediments and other tracers across ocean basins [Krumpen *et al.*, 2019]. With climate change, the area and thickness of sea ice is reduced in the Arctic, which leads to a more mobile sea-ice cover and positive trends in sea-ice velocity [Kwok *et al.*, 2013, Spreen *et al.*, 2011]. Trends in sea-ice motion, linked to trends in wind speed, are also observed in the Southern Hemisphere (SH) [Holland and Kwok, 2012, Kwok *et al.*, 2017].

Satellite remote sensing has developed as an attractive option to monitor sea-ice drift consistently across the polar sea-ice cover at a daily to sub-daily frequency. The initial work by Ninnis *et al.* [1986] was followed by many investigators using a variety of satellite imaging sensor technologies as input, including visible and infrared radiometry [Emery *et al.*, 1991], microwave radiometry and scatterometry [Agnew *et al.*, 1997, Girard-Ardhuin and Ezraty, 2012, Kwok *et al.*, 1998, Lavergne *et al.*, 2010, Liu *et al.*, 1999], and synthetic aperture radar (SAR) [Komarov and Barber, 2014, Kwok *et al.*, 1990, Muckenhuber *et al.*, 2016]. The various imaging technologies however lead to sea-ice motion fields with different characteristics, e.g. medium spatial resolution (~ 20 km) and coverage limited by cloud cover for the visible and infrared radiometry, high spatial resolution (~ 5–10 km) but coverage limited by acquisition repeat cycles for the SAR imagery, and coarse spatial resolution (> 30 km) and daily complete coverage for the microwave radiometers and scatterometers. Despite the imaging technologies being very different from each other, the motion tracking algorithms employed are quite similar and stem from the maximum cross-correlation (MCC) technique [Emery *et al.*, 1986].

Note: The characteristics of the CIMR mission, and especially the spatial resolution obtained at K (5 km) and Ka (4 km) should allow global, year-round (irrespective of solar illumination and cloud cover), sub-daily moniroting at medium spatial resolution (25 km), which is unprecedented.

The Level-2 sea-ice drift product described in this ATBD has been specifically designed to fully exploit the CIMR mission, in particular:

1. the CIMR sea-ice drift product is a Level-2 product, adopting a swath-to-swath approach [Lavergne *et al.*, 2021]. This is conversely to all existing sea-ice drift products from passive microwave missions (OSI SAF, IFREMER, NSIDC, etc…) that are Level-3 products (available once a day from daily averaged maps of brightness temperatures). SAR-based products (e.g. from the CMEMS Sea Ice Thematic Assembly Center) are generally swath-to-swath products (but with spatial coverage limited by the coverage of the SAR missions).
2. the CIMR sea-ice drift product aims at a 25 km grid spacing, taking full advantage of the K and Ka imagery.
3. sea-ice drift is derived both from the forward and backward scans separately to improve quality control (filtering of “rogue” vectors)

CHAPTER

SIX

BACKGROUND AND JUSTIFICATION OF SELECTED ALGORITHM

6.1 The Continuous Maximum Cross-Correlation method (CMCC)

As for many other motion tracking methods applied in geophysics [Maslanik *et al.*, 1998, Notarstefano *et al.*, 2007], sea ice drift is tracked from a pair of images, with a block-based strategy. Each block (*feature*, *sub-image*, …) is composed of a limited ensemble of pixels from the first image (the *reference* block) and centred at a tracking location, for which the most similar block in the second image (the *candidate* block) is looked for. The degree of similarity is assessed by a metric, which often is the correlation coefficient between the reference and candidate blocks. The maximum correlation indicates the best match and the two-dimensional offset between the centre points of the two blocks is the drift vector.

The description given above applies to the well known Maximum Cross Correlation (MCC) technique which has been successfully applied by many investigators [Emery *et al.*, 1991, Haarpaintner, 2006, Kwok *et al.*, 1998, Ninnis *et al.*, 1986, Notarstefano *et al.*, 2007, Schmetz *et al.*, 1993], among others). In the MCC, wthe search for the best candidate block is *discrete* and *exhaustive*. Discrete since the offsets between the centre points are in whole number of pixels and exhaustive because all candidate blocks are evaluated before the best can be chosen.

The same description applies to the Continuous Maximum Cross Correlation technique (CMCC) method introduced by Lavergne *et al.* [2010]. CMCC is the strategy developed and implemented first for the low resolution sea ice drift product of the EUMETSAT OSI SAF. Conversely to the MCC, the search for the best candidate vector is performed in a continuous manner over the two-dimensional plane and, as a consequence, the search algorithm is not exhaustive.

Although more complex to implement and, by nature, potentially less robust than the MCC technique, the CMCC has the advantage of removing the *quantization noise* which has hindered the retrieval of smooth motion vector fields when the time span between the images is shortened [Ezraty *et al.*, 2007, Haarpaintner, 2006, Kwok *et al.*, 1998]. **This is particularly important when the CIMR Level-2 product aims at sub-daily drift detection.**

6.2 Choice of K and Ka as main microwave frequencies

As introduced above, sea-ice motion tracking from pairs of satellite images does not require specific microwave frequencies to work. As long as the frequency allows a mostly unperturbed view of the sea-ice surface (window channels), the accuracy of the sea-ice drift retrieval does not depend on the frequency per se, but rather on the spatial resolution achieved. Imagery with higher spatial resolution should result in higher accuracy of the motion vectors.

This is the reason why the primary microwave frequencies for the Level-2 sea-ice drift product are the K (18.7 GHz) and Ka (36.5 GHz) channels that will achieve spatial resolution of 5 km and better. Experience from the EUMETSAT OSI SAF near-real-time and climate processing confirms that the Ka-band imagery of AMSR2 provides the best results. In CIMR, the spatial resolution of the K-band imagery will be only slightly worse than at Ka-band, so that it should be included in the main processing. Kwok [2008] showed that the K-band imagery of AMSR2 provided valuable sea-ice motion information during the summer melt season, but Lavergne *et al.* [2021] did not find better performance of summer sea-ice motion retrievals from K compared with Ka.

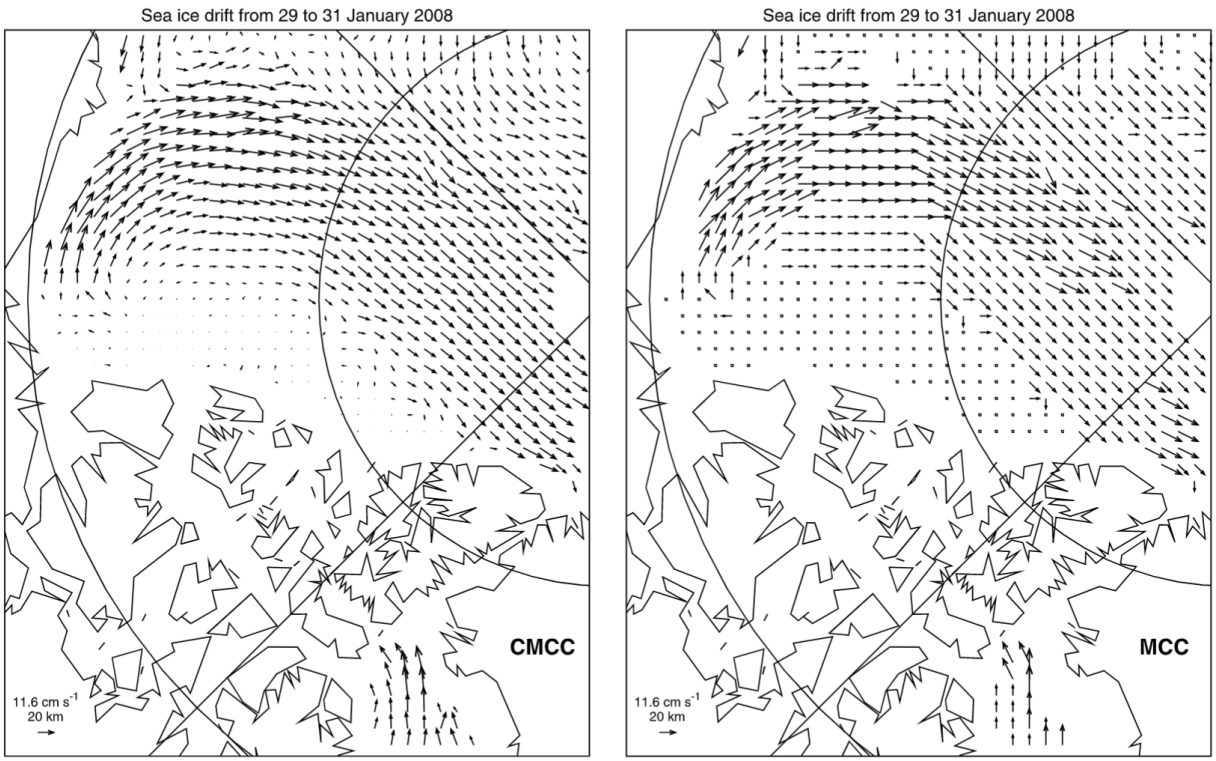


Fig. 6.1: Example se-ice displacements from AMSR-E (37 GHz H and V channels) over the Beaufort Sea and Canadian Basin from 29 to 31 January 2008. The product was processed with the (left) Continuous Maximum Cross Correlation (CMCC) and (right) Maximum Cross Correlation (MCC) method from the same satellite images. On the MCC product, zero-length vectors are depicted with a small square symbol while tiny arrows are used for the CMCC product. The MCC product clearly exhibits a quantization noise. Such an artifact is not present in the CMCC drift data set (reproduced from [Lavergne *et al.*, 2010]).

The lower frequency channels of CIMR (L, C, X) could possibly contribute to sea-ice motion tracking, but two challenges are their coarser spatial resolution and the lack of intensity patterns to track from one image to the next (K and Ka imagery are sensitive to snow and sea-ice type, which has large variability across the polar sea-ice and is stable over the tracking window of 0,5 to 2 days). This ATBD thus focus on K and Ka imagery but leaves the door open for later inclusion of at least C and X.

6.3 Exploitation of the forward and backward scan

In the past (e.g. Girard-Ardhuin and Ezraty [2012], Haarpaintner [2006], Kwok *et al.* [1998], among others), one sea-ice motion vector field would have been processed for each microwave channel as input, i.e. one vector field from the K-H band, one from K-V, etc… the different vector fields would then be merged together a-posteriori.

Lavergne *et al.* [2010] introduced a different approach where a single sea-ice drift motion field is processed in one go from all the available imagery channels. This *implicit* merging it implemented at the core of the motion tracking algorithm, by solving for the maximum value of the sum of the cross-correlation of several imagery channels, instead of just one imagery channel.

For CIMR, this can be further extended by considering the imagery from *forward* and *backward* scans as independent imaging channels. There are thus two images with K-H, two with K-V, etc… in total eight imaging channels for each swath. When doing sea-ice motion tracking with CIMR, one can thus do an implicit merging with 16 pairs of images (fwd-fwd, fwd-bck, bck-fwd, and fwd-bck) for each of K-V, K-H, Ka-V, and Ka-H. It is expected that using the forward and backward scans as part of the implicit merging will be benefitial to reduce the retrieval uncertainty and limit the number of rogue vectors. This will have to be validated in the product development phase since CIMR is the first passive microwave mission offering full scans for sea-ice motion tracking.

6.4 Swath-to-swath Motion Tracking (Level-2 strategy)

One of the key characteristics of the CIMR Level-2 sea-ice drift product is that it will be a "swath-to-swath" product, thus computed at the intersection of individual swaths. Fig. 6.2 illustrates the concept.

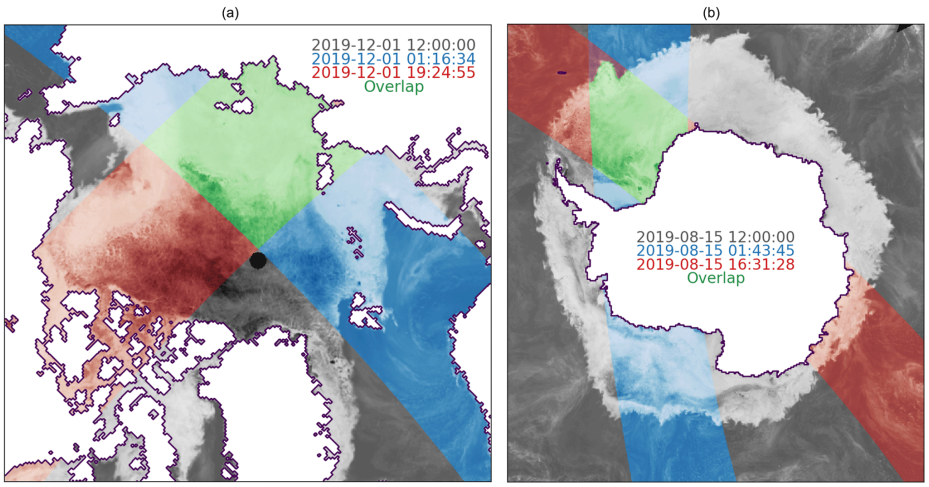


Fig. 6.2: (a) Daily average map of AMSR2 36.5 GHz V-pol TB on 1 December 2019 (greys) for the Arctic and two individual gridded swaths on the same day (blues: 01:16:34 UTC; reds: 19:24:55 UTC). The sea-ice region of overlap between the two swaths is highlighted in greens and is where S2S drift vectors can be computed. (b) Similar but for the Antarctic on 15 August 2019 (blues: 01:43:45 UTC; reds: 16:31:28 UTC). Reproduced from [Lavergne *et al.*, 2021].

The advantages of choosing a swath-to-swath approach (instead of a daily L3 product) are presented in Lavergne *et al.* [2021]. We reproduce their conclusions below:

> We investigate the feasibility and impact of adopting a swath-to-swath (S2S) vs. daily map (DM) framework for the processing of sea-ice motion from modern passive microwave satellite missions such as JAXA's AMSR2 in preparation for the future CIMR mission. We find that S2S sea-ice drift vectors obtained from AMSR2 imagery are more accurate than the corresponding DM vectors when compared to GPS trajectories from on-ice buoys in both the Arctic and Antarctic. An S2S configuration also results in many more drift vectors on a daily basis: the number varies with latitude and depends on the orbital and swath characteristics of the satellite mission. Since S2S drift vectors can be prepared for each new incoming swath, this configuration yields much better timeliness, which is beneficial for several operational applications such as support to navigation and short-term sea-ice forecasting. One potential limitation to the S2S configuration is that it is more sensitive to inaccurate geolocation, especially if the geolocation errors are systematic (e.g. a shift in the flight direction).
>
> As far the CIMR mission is concerned, we recommend the adoption of an S2S configuration for the Level-2 sea-ice drift product in the operational ground segment. Considering the microwave frequency channels, target spatial resolution, swath width and geolocation accuracy specified for the CIMR imagery, its Level-2 sea-ice drift product will allow for unprecedented spatial resolution, coverage and accuracy for a microwave radiometer mission. Several other new characteristics of the CIMR mission (e.g. the relatively high spatial resolution at 6.9 and 10.8 GHz, the backward and forward scans) will also contribute to an enhanced sea-ice drift product, but this requires further research.

CHAPTER

SEVEN

LEVEL-2 PRODUCT DEFINITION

Table 7.1, Table 7.2, Table 7.3 define the content (TBC) of the Level-2 sea-ice drift product files. Their structure is inspired by that of the EUMETSAT OSI SAF products.

Table 7.1: NetCDF Group: Processed data (TBC)

name	description	units	dimensions
crs	Definition of the EASE2 product grid	n/a	n/a
dX	Component of the drift along the X-axis of the EASE2 grid	m	(nx,ny)
dY	Component of the drift along the Y-axis of the EASE2 grid	m	(nx,ny)
lat_1	Latitude of the end point of the drift vector	degrees North	(nx,ny)
lon_1	Longitude of the end point of the drift vector	degrees East	(nx,ny)
sX	Uncertainty (one standard deviation) of the dX component	m	(nx,ny)
sY	Uncertainty (one standard deviation) of the dY component	m	(nx,ny)
cXY	Correlation between the uncertainty in dX and in dY	m	(nx,ny)
status_flag	A flag indicating status of retrieval, e.g. “nominal”, “over land”, “close to ice edge”, “over ocean”, “corrected from rogue vector”,…	n/a	(nx,ny)

Note: The *CIMR* Level-2 *Sea Ice Drift* product is an Earth-gridded product. The dimensions of each variable in the Level-2 file refer to the (nx,ny) dimensions of the product grid in an *EASE2* polar projection (see Table 8.1).

Each data variable in the `Processed data` group holds conventional attributes following CF-1.7 or above:

Table 7.2: Standard variable attributes (TBC)

name	description
standard_name	A standard name that references a description of a variable”s content
long_name	A descriptive name that indicates a variable”s content
_FillValue	value that will appear in the dataset when the data is either missing or undefined
units	unit of measure

Table 7.3: Some global attributes in the Level-2 product files (TBC)

name	description
title	CIMR L2 Sea Ice Drift product
processing_level	Level-2
time_coverage_start	valid *start* time of the product
time_coverage_end	valid *end* time of the product
area	(Northern or Southern) Hemisphere
TBD	TBD

CHAPTER

EIGHT

BASELINE ALGORITHM DEFINITION

In the following sections, the Level-2 sea-ice drift algorithm is further described. It consists in those steps:

- resampling of Level-1b data;
- selection of tracking locations and preliminary screening;
- block-based maximization of the correlation metric via the CMCC;
- filtering and correction step;
- assign status flags and per-vector uncertainties.

Sea-ice motion tracking is performed at the intersection of two swaths. The motion tracking itself requires that the brightness temperature imagery is first resampled to a common grid. This makes the level-2 sea-ice drift product quite different from other level-2 products in that:

1. the Level-1b data must be remapped to a common (polar) grid as part of the processing, e.g. an EASE2 grid;
2. as a consequence, the Level-2 sea-ice drift product is presented on an EASE2 grid (not in a swath projection);
3. the algorithm operates not with one, but with at least **two** Level-1b files as input;
4. for each incoming Level-1b file, one can generate more than one Level-2 sea-ice drift product (consider the swath intersection with the most recent Level-1b swath, the one before, etc…)

8.1 CIMR Level-1b re-sampling approach

The re-sampling approach for resampling CIMR Level-1b K and Ka band imagery is not defined at this stage. From experience with sea-ice motion tracking from other passive microwave mission, the Level-1b re-sampling approach does not have a large influence on the results. At this stage, the following characteristics are expected from the re-sampling:

1. Remap incoming Level-1b files on two EASE2 polar grids (one covering Northern Hemisphere, the other covering the Southern Hemisphere);
2. Remap 4 imagery channels (K-V, K-H, Ka-V, Ka-H);
3. Remap the forward and backward scans separately;
4. Aim at a grid spacing close to 5 km (TBC).

Table 8.1 defines four grids, two for the Northern Hemisphere, two for the Southern Hemisphere. nx (ny) is the number of grid cells in x (y) dimension, Ax (Ay) is the grid spacing, and Cx (Cy) is the coordinate of the upper-left corner of the upper-left cell in the grid. Grids `(n,s)h_ease2-005` have 5 km grid spacing and are candidate target grids on which to remap the Level-1b imagery. Grids `(n,s)h_ease2-250` have 25 km grid spacing and are candidate grids for the resulting Level-2 sea-ice drift product. By construction, the center of the cells of the two '250' grids fall exactly at the center of one every five grid cells of the '005' grids. This ensures that drift vectors (at every grid cell of the '250' grids) use image blocks (from the '005' grids) that are perfectly aligned.

Table 8.1: Definition of EASE2 grids used in the Level-2 sea-ice drift processing (TBC)

Id	nx	ny	Ax [km]	Ay [km]	Cx [km]	Cy [km]
`nh_ease2-250`, `sh_ease2-250`	432	432	25.0	25.0	-5400.0	-5400.0
`nh_ease2-005`, `sh_ease2-005`	2160	2160	5.0	5.0	-5400.0	-5400.0

Such a '1-every-5' construct is not a requirement of the sea-ice drift algorithm, but a simplification used in many motion extraction algorithms. Other alternatives are to grid the Level-1b data with 4 km spacing and consider a '1-every-6' ratio (Level-2 grid spacing at 24 km) or keep a '1-every-5' ratio (Level-2 grid spacing at 20 km).

The remaining of the algorithm description does not depend on this choice, which can easily be changed in the course of the product development.

Each incoming Level-1b file thus results in 16 gridded fields of brightness temperature: 2 hemispheres x 2 scans x 4 bands. These can be written in two L1C-like netCDF files (1 per hemisphere) or directly entered the motion tracking algorithm. They must be written to netCDF files at one point so that they are available for sea-ice drift processing when the next Level-1b swath file arrives.

8.2 Algorithm Assumptions and Simplifications

All block-based motion have similar assumptions and simplifications. They assume that each pixel in a block moves at a constant rate from one image to the next: what is retrieved are the `dx` and `dy` components of the motion vector. In case of rotational motion within the area of the image block, the retrieved components will be those representing most faithfully the change in intensity between the two images, but the rotation rate is not measured. By the same token, if deformation (convergence / divergence / shear) occurs within the area of the image blocks, this will not be detected. Rotation and deformation between image blocks (between neighboring pixels) can of course be detected.

To detect and possibly correct "rogue" vectors in the motion field, we have to assume that the motion field is spatially coherent in a neighborhood. This is because we detect anomalous motion vectors by their distance to the local average motion. One must be careful with this assumption as to not artificially smooth the motion fields and remove actual diverging / converging motion.

8.3 Level-2 end to end algorithm functional flow diagram

Fig. 8.1 shows the flow diagram for the CIMR Level-2 sea-ice drift algorithm. Note the structure in two chains: preprocessing of the image, and sea-ice motion tracking per se.

8.4 Functional description of each Algorithm step

8.4.1 Pre-processing

Prepare ice/ocean/land mask

The Level-2 sea-ice drift algorithm shall only be applied over sea-ice portions of the image. We thus need to prepare an ice/ocean mask as well as a land-mask to only process over sea ice.

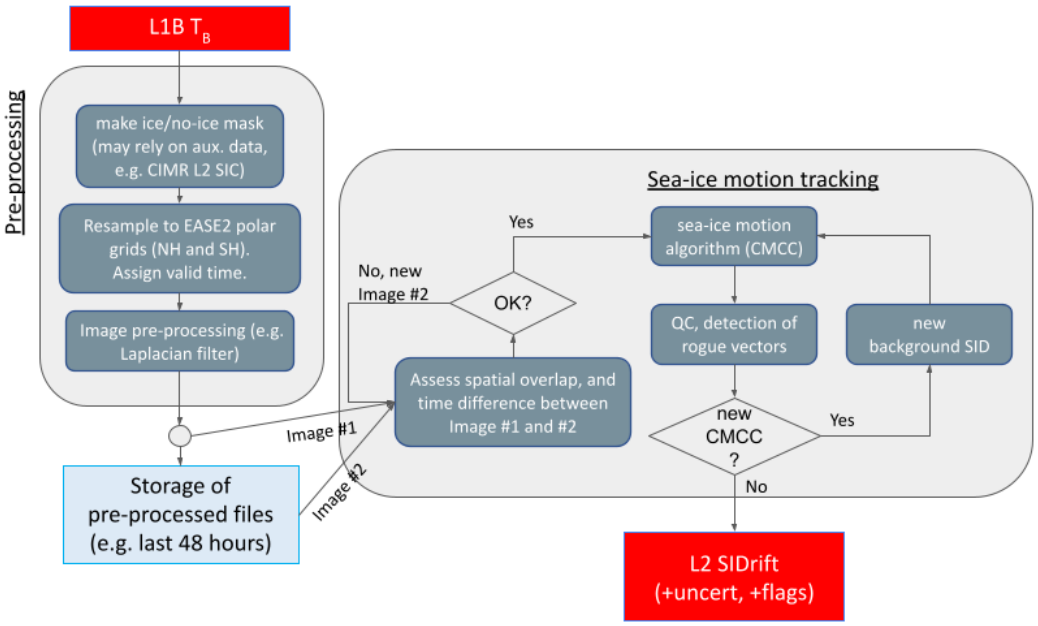


Fig. 8.1: End-to-end algorithm flow diagram of the CIMR Level-2 sea-ice drift algorithm

Input data

The latitude and longitude of the image grids `nh_ease2-005` and `sh_ease2-005`.

Output data

A 2D field of ice/no-ice/land on the image grids.

Auxiliary data

A source of sea-ice concentration (can be from CIMR Level-2 Sea Ice Concentration product, or Level-2 Integrated-Retrieval product).

Remap Level-1b brightness temperatures to the EASE2 image grids

The Level-1b data (brightness temperature at K-V, K-H, Ka-V, Ka-H) are remapped onto the EASE2 image grids `nh_ease2-005` and `sh_ease2-005`. The forward and backward scans are remapped independently.

A *valid time* is associated to the remapped imagery. It can be defined as the mean time of all the Level-1b samples remapped onto the image grid or the time of the observation closest to the (North or South Pole). The exact definition is not critical, neither is the accuracy, so that a single valid time is associated to all 8 remapped imagery bands in the Northern Hemisphere, and a single (different) time to the 8 remapped imagery bands in the Southern Hemisphere.

Mathematical description

Remapping is the process of computing the Level-1b brightness temperature onto a regular Earth-based grid. The values in the image grid can generally be described as a weighted sum of the values in the Level-1b swath projection. The mathematical description is TBD and will depend on the selected remapping strategy.

Input data

The latitude and longitude of the image grids `nh_ease2-005` and `sh_ease2-005`.

The Level-1b brightness temperatures of all samples in the input Level-1b file, split in a forward and a backward scan.

Output data

16 2D field of brightness temperature on the image grids (2 grids x 2 scans x 4 channels).

Ancillary data

The latitude and longitude of all the Level-1b samples.

Image preprocessing (Laplacian filter)

The remapped brightness temperatures are not used directly as input to the sea-ice motion tracking step. Instead, a filter is first applied to the remapped imagery to enhance and stabilize intensity patterns.

At the end of the step, the Laplacian processed imagery are saved to disk, e.g. in netCDF files (two separated files for the two hemispheres).

Mathematical description

A common filter, used by Girard-Ardhuin and Ezraty [2012] and Lavergne *et al.* [2010], is the Laplace filter, that is based on computing the second derivatives of the image intensity.

$$\begin{aligned}\mathcal{L}[i,j] = &\frac{1}{N^+}\sum_{n=i-1}^{i+1}\sum_{m=j-1}^{j+1}\delta_{\text{na}}(n,m)\delta_{\text{ice}}(n,m)\delta_1^{i,j}(n,m)\mathcal{I}[n,m] \\ &-\frac{1}{N^-}\sum_{n=i-2}^{i+2}\sum_{m=j-2}^{j+2}\delta_{\text{na}}(n,m)\delta_{\text{ice}}(n,m)\delta_2^{i,j}(n,m)\mathcal{I}[n,m]\end{aligned} \tag{8.1}$$

with

$$\delta_k^{i,j}(n,m) = \begin{cases} 1 & \text{if } |i-n| = k \text{ or } |j-m| = k; \\ 0 & \text{otherwise.}\end{cases}$$

$$N^+ = \sum_{n=i-1}^{i+1}\sum_{m=j-1}^{j+1}\delta_{\text{na}}(n,m)\delta_{\text{ice}}(n,m)\delta_1^{i,j}(n,m) \leq 8$$

$$N^- = \sum_{n=i-2}^{i+2}\sum_{m=j-2}^{j+2}\delta_{\text{na}}(n,m)\delta_{\text{ice}}(n,m)\delta_2^{i,j}(n,m) \leq 16$$

In Eq. (8.1) and (8.2), $\delta_{\text{na}}(n,m)$ has value 0 if $\mathcal{I}[n,m]$ is non-available (a missing value in the swath, or missing values outside the coverage of the swath) and $\delta_{\text{ice}}(n,m)$ is 1 only over ice pixels, as specified by the ice/water/land mask. It means that only *valid*, *sea ice* pixels enter the Laplacian field in order to limit spurious features along the ice edge, coastline or at the border of the swath coverage.

$\mathcal{L}[i,j]$ is only computed if the centre cell $[i,j]$ is itself over sea ice, $\delta_{\text{ice}}(i,j) = 1$.

In the event when $N^+ < 5$ or $N^- < 9$, not enough pixels are available for computing $\mathcal{L}$ and a missing value is stored at grid cell $[i,j]$.

Input data

The remapped brightness temperatures on the image grids (16 2D fields).

Output data

16 2D field of laplacian-filtered brightness temperature on the image grids.

Ancillary data

The ice/water/land mask on the image grids.

8.4.2 Sea-ice motion tracking

To compute sea-ice drift vectors require two images. A *start* and an *end* image. In the near-real-time Level-2 processing context, the end image is the (Laplacian filtered) remapped imagery from the input Level-1b file. Start images are taken from a running pool of remapped imagery from the previous runs of the sea-ice drift algorithm. In principle, many start images can be selected and run into the sea-ice motion tracking against the end image. The difference between the valid time of the start and end imagery determines the drift duration (*aka* time span) of the Level-2 drift vectors.

At minimum (and in priority), the Level-2 sea-ice drift algorithm should be applied once with a start image with valid time approximately 24 hours before the valid time of the end image. This results in 24 hours drift vectors. Ideally, the sea-ice motion tracking algorithm described below is applied for several (start, end) image pair to obtain a good temporal sampling of the sea-ice motion (e.g. 24 hours, 18 hours, 12 hours, 6 hours, etc…). At max, the sea-ice motion tracking algorithm would be applied with all (start, end) image pairs for which the valid time of the start image is less or equal to 24 hours from the valid time of the end image. Each (start, end) image pair will correspond to different area of intersect between the two swaths, and thus to different numbers of resulting sea-ice drift vectors. Pairs with too limited overlaps could be discarded up-front to favor processing pairs with a large overlap. Because swath-to-swath motion tracking is very sensitive to (systematic) geolocation errors [Lavergne *et al.*, 2021] it might be preferable to not process some pairs (e.g. ascending vs descending) that would have larger uncertainties.

A hard limit is that the processing of all the image pairs must happen before the next input Level-1b input file is available for processing. Parallel computing strategy will help reduce the total processing time (since each image pair can be processed independently of each others). A scheduling strategy must be designed to afford the maximum number of image pairs before it causes a problem for product latency.

At this stage it is TBD if each (start, end) image pairs results in individual Level-2 product files (better for the latency) or if all the sea-ice drift vectors (with different time spans) are concatenated in a single Level-2 product file.

In any case, the description below is for a single (start, end) image pair, and for each image grid (nh and sh) separately.

Select grid locations where to apply the CMCC algorithm

The CMCC can only be applied where the two swaths (start and end images) overlap over sea ice. A first step in the processing is thus to go through all the positions in the `(n,s)h_ease2-250` grid and test if there is sufficient overlap between the two images, and if the overlap is over sea ice.

Logical description

For each grid position in the Level-2 product grid, the following tests are performed.

Masking of land pixel (step 1)

- Blocks whose centre pixel is over land are discarded;

Masking of pixels with not enough sea ice (step 2)

- The start and end blocks of the two ice/water/land masks are loaded. Discard the grid locations whose start and end blocks are not entirely over ice.

Masking of pixels with missing data (step 3)

- The start and end blocks of the Laplacian images are loaded. Discard the grid locations whose blocks hold missing data.

The positions that are not discarded after those three steps are passed to the next processing step (CMCC).

Input data

The two (start and end) ice/ocean/land masks on the image grid `(n,s)h_ease2-005`.

The two (start and end) Laplacian-filtered brightness temperature maps on the image grid `(n,s)h_ease2-005`.

The diameter of the sub-image (*aka* image block) to be used in the CMCC, in number of pixels.

The definition of the Level-2 product grid `(n,s)h_ease2-025`.

Output data

A 2D field of status flags recording the status at the end of this selection step (in particular recording where and why CMCC will not be attempted).

Run the CMCC algorithm

The CMCC is the core sea-ice motion algorithm

Mathematical description

We note $\mathcal{L}_0(x,y)[i]$ the i^{th} pixel of the *start* sub-image centered at point (x,y), extracted from the $\mathcal{L}_0$ image. (x,y) are the coordinates expressed in the underpinning EASE2 projection (units km).

The total number of pixels N in a sub-image depends on the diameter of the sub-image (input parameter).

The mean and standard deviation values for a given sub-image are:

$$\begin{aligned}\langle\mathcal{L}_0(x,y)\rangle = &\quad \frac{1}{N}\sum_{i=1}^{N}\mathcal{L}_0(x,y)[i]\\ \sigma(\mathcal{L}_0(x,y)) = &\quad \sqrt{\langle\mathcal{L}_0^2(x,y)\rangle - \langle\mathcal{L}_0(x,y)\rangle^2}\end{aligned}$$

Similar values can be computed for a *end* sub-image centred at (u,v): $\langle\mathcal{L}_1(u,v)\rangle$ and $\sigma(\mathcal{L}_1(u,v))$.

The match between a start and a stop sub-image is evaluated via the correlation metric:

$$\rho(x,y,\delta_x,\delta_y) = \frac{\sum_{i=1}^{N}(\mathcal{L}_0(x,y)[i] - \langle\mathcal{L}_0(x,y)\rangle)(\mathcal{L}_1(x+\delta_x,y+\delta_y)[i] - \langle\mathcal{L}_1(x+\delta_x,y+\delta_y)\rangle)}{\sigma(\mathcal{L}_0(x,y))\sigma(\mathcal{L}_1(x+\delta_x,y+\delta_y))} \tag{8.2}$$

By construction, $\rho(x,y,\delta_x,\delta_y)$ takes values between -1 and $+1$. High values indicate a good match between the sub-images. This is further interpreted as having found the offsets $\delta_x = u - x$ and $\delta_y = v - y$ which best explain the local change in intensity between the two sub-images. (δ_x,δ_y) is the drift vector.

Pixels of the candidate block $\mathcal{L}_1(x+\delta_x,y+\delta_y)$ are computed from bi-linear interpolations of the pixels of $\mathcal{L}_1$. For example, $\mathcal{L}_1(u,v)[i]$ is given by:

$$\begin{aligned}\mathcal{L}_1(u,v)[i] = &(1-\epsilon_u)\times(1-\epsilon_v)\times\mathcal{L}_1(\bar{u},\bar{v})[i]\\ &+(1-\epsilon_u)\times\epsilon_v\times\mathcal{L}_1(\bar{u},\bar{v}+s_v)[i]\\ &+\epsilon_u\times(1-\epsilon_v)\times\mathcal{L}_1(\bar{u}+s_u,\bar{v})[i]\\ &+\epsilon_u\times\epsilon_v\times\mathcal{L}_1(\bar{u}+s_u,\bar{v}+s_v)[i]\end{aligned} \tag{8.3}$$

where

$$\begin{aligned}\bar{t} = &\quad \text{Trunc}(t)\\ \epsilon_t = &\quad |t-\bar{t}|\\ s_t = &\quad \frac{t}{|t|}\end{aligned}$$

For example, for $t=-2.8$, $\bar{t}=-2$, $\epsilon_t=0.8$ and $s_t=-1$. Eq. (8.3) permits computing *virtual* sub-images at continuously varying centre points (u,v) and thus building a continuous optimization framework to the estimation of motion vectors from a pair of images.

Finding the motion vector (δ_x,δ_y) at position (x,y) can be expressed as the following maximization problem:

$$\max_{(x,y)\in\mathcal{D}}\rho(x,y,\delta_x,\delta_y) \tag{8.4}$$

which is solved at all grid positions where the motion vector is searched for (see previous step). Each optimization is conducted independently from the others. $\mathcal{D}$ is a validity domain for (δ_x,δ_y). Eq. (8.4) thus defines a two dimensional optimization problem with domain constraint.

Eq. (8.4) is valid for one pair of images. In the CIMR sea-ice drift algorithm, we however envision not one pair of (start, end) images but 16 pairs (fwd-fwd, fwd-bck, bck-fwd, and fwd-bck) for each of K-V, K-H, Ka-V, and Ka-H considering the foward and backward scans as separate images. Following Lavergne *et al.* [2010] we implement an inplicit merging of the information content of the 16 imaging channels by maximizing a sum of cross-correlation functions:

$$\max_{(x,y)\in\mathcal{D}}\frac{1}{N_{ch}}\sum_{c=1}^{c=N_{ch}}\rho^c(x,y,\delta_x,\delta_y) \tag{8.5}$$

where N_{ch} is the number of channels ($N_{ch} = 16$).

Eq. (8.5) is solved by the Nelder-Mead algorithm [Nelder and Mead, 1968]. This algorithm is chosen since it is simple to implement and does not require computing the gradients of the function to be minimized. It furthermore has good convergence and computational properties in problems with low dimensionality [Lagarias *et al.*, 1998].

Starting points for the optimization are sampled on a length-angle regular grid around point $(0, 0)$ as on Fig. 8.2.

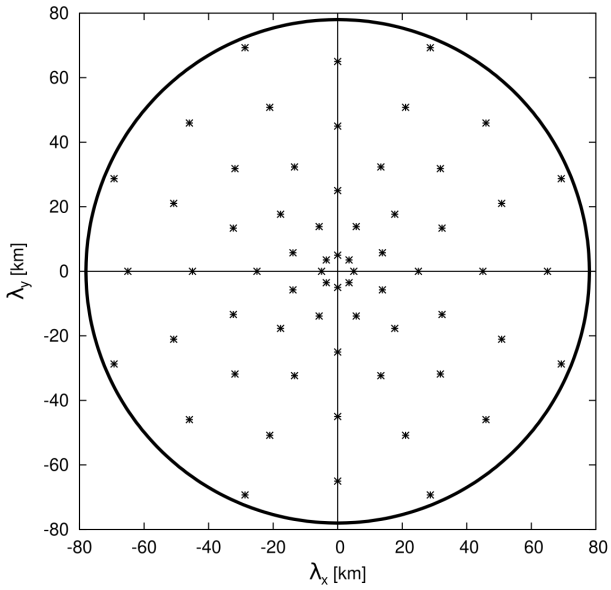


Fig. 8.2: Initialization points for the Nelder-Mead algorithm, chosen on a regular length-angle grid aroung point $(0, 0)$.

The length increment is set to 10km and the angular increment to $45°$. The circle has radius $\mathbf{L}$, the maximum drift distance defining $\mathcal{D}$.

$\rho(x, y, \delta_x, \delta_y)$ is computed at each of those points and the best 3 vertexes are kept for initialising the Nelder-Mead optimization.

Termination and convergence is tested upon via a relative difference of function values at the current *best* and *worst* vertexes, f_b and f_w. Specifically, the algorithm is said to have converged if and only if $|f_b - f_w| < (f_b + f_w) \times \tau + \epsilon$, with τ and ϵ small and positive floating point values. As a safeguard, the maximum number of iterations is set to 1000.

In Eq. (8.5) $\mathcal{D}$ is a disc shaped domain expressing the *a-priori* knowledge we bring to the optimization problem. Its purpose is to limit the search area for the solution vector during the optimization process. It is defined by a centre point (x_c, y_c) and radius $\mathbf{L}$.

$$(\delta_x, \delta_y) \in \mathcal{D}_{x_c, y_c} \Leftrightarrow d(x_c, y_c; \delta_x, \delta_y) < \mathbf{L} \tag{8.6}$$

In Eq. (8.6), $d(x_c, y_c; \delta_x, \delta_y)$ is the distance (great circle) between the centre point of $\mathcal{D}$ and the tip of the drift vector (δ_x, δ_y). (x_c, y_c) represents our best *a-priori* knowledge at the time of performing the optimization. It is initially set to $(0, 0)$.

Eq. (8.6) cannot be used *as is* in the optimization routine since it leads to abrupt and non-linear behavior. $\mathcal{D}$ is instead implemented as a *soft* constraint based on a mono-dimensional sigmoid function $W(d)$:

$$W(d) = \frac{1}{1 + e^{k(d - \mathbf{L})}} \tag{8.7}$$

In Eq. (8.7), k is a parameter controlling the steepness of the sigmoid around the cut-off value $\mathbf{L}$. By construction, $W(\mathbf{L}) = 0.5$. By using a large enough value for k, the W can be made arbitrarily close to the Heaviside step function, yet remaining smooth and continuous.

Eq. (8.8) illustrate how the penalty is applied to the correlation function $\rho^c(x, y, \delta_x, \delta_y)$ (Eq. (8.5)).

$$\rho_D^c(x, y, \delta_x, \delta_y) = (\rho^c(x, y, \delta_x, \delta_y) + 1) \times W(d(x_c, y_c; \delta_x, \delta_y)) - 1 \tag{8.8}$$

Fig. 8.3 plots a mono-dimensional example of applying a sigmoid penalty function to a synthetic correlation function. Evaluations for x lower than L are dominated by the correlation value $\rho(x)$ while those occurring outside the domain (x larger than L) return very bad scores, that is close to -1.

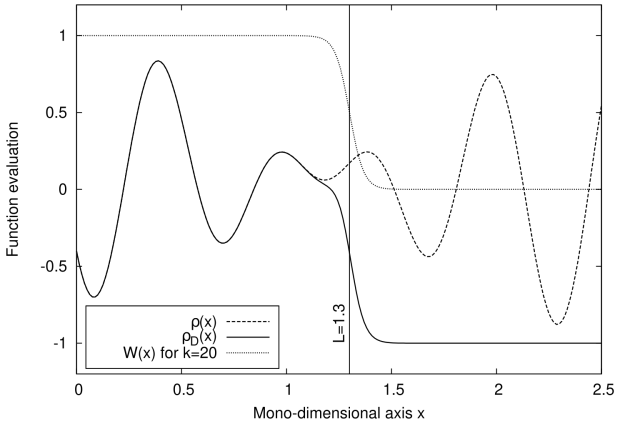


Fig. 8.3: Example soft constraint implemented with a sigmoid penalization function W and its application on a synthetic, mono-dimensional correlation signal ρ. Here, the **L** parameter is 1.3 and k is 20.}

In Eq. (8.8), ρ_D is the penalised correlation function. Finding the maximum of ρ_D is taken as a proxy for solving the original, constrained, optimization problem of Eq. (8.5). ρ_D is the function entering the Nelder-Mead algorithm instead of ρ.

It is customary to compute **L** as a maximum expected speed v_{max}, multiplied by the time separation between the valid times of the two images $\mathbf{T}_1$ - $\mathbf{T}_0$. **L** is thus the maximum expected straight-line distance that can be covered in the given time.

Input data

The diameter of the sub-image (*aka* image block) to be used in the CMCC, in number of pixels.

The maximum allowed drift speed v_{max} (to define **L**) and \mathcal{D}).

The 8 imagery bands for the start and end images, as well as the associated valid times **T**.

Output data

A 2D field of drift vectors (δ_x and δ_y components) on the `(n,s)h_ease2-250` grid.

A 2D field of maximum cross-correlation value for each vector.

A 2D field of status flags recording the status at the end of the CMCC optimization.

Quality Control: detect and correct outliers ("rogue" vectors)

Once the CMCC described above has been applied once to each of the start positions selected by the preliminary checks, a filtering step is taken to detect, correct or remove obviously erroneous vectors (so called "rogue" vectors).

Causes for those erroneous vectors include:

1. convergence of the Nelder Mead algorithm in a local maximum;
2. noise in the sub-images;
3. edge effects in the sub-images.

Whatever the reason be, the filtering step is based on the distance from individual displacement vectors to the average of its neighboring vectors. If this distance is less than a fixed threshold, the displacement vector being tested is validated and another vector is tested upon. Otherwise, a new CMCC motion tracking optimization is triggered.

In this new CMCC optimization, the Nelder Mead algorithm is initialised and run like in the previous section, except that the validity domain **D** is adapted (center and radius) to translate the new constraint.

Erroneous vectors are detected and corrected one by one, from the "most erroneous" (see below) until all vectors are either corrected or flagged as bad.

Mathematical description

Let Δ_{avg} be the distance between the tip of the current drift vector (δ_x, δ_y) and the tip of the zonal average drift vector $(\delta_x^{\text{avg}}, \delta_y^{\text{avg}})$. The average drift vector is computed from the 8 neighboring drift vectors, that is the 8 closest vectors *not including the current one*. The local **D** domain is then the disc with centre $(\delta_x^{\text{avg}}, \delta_y^{\text{avg}})$ and radius $\Delta_{\text{max}}^{\text{avg}}$. $\Delta_{\text{max}}^{\text{avg}}$ is set to 10km. Neighboring vectors with a maximum correlation value of less than 0.5 are not used, to avoid degrading the average drift field with possibly wrong estimates.

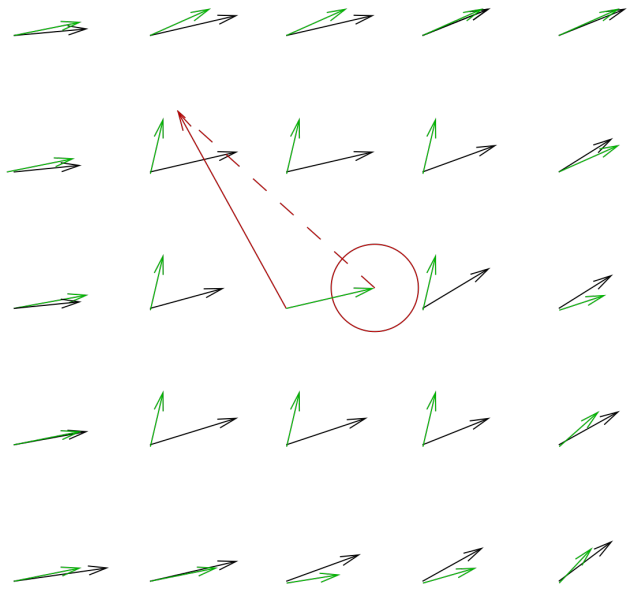


Fig. 8.4: Example case where the current drift vector (in red) is obviously erroneous, considered the smoother vector field from the first estimate from CMCC (in black). The locally averaged vector field is plotted in green. Δ^{avg} is the length of the dashed red line. The red disc has radius $\Delta_{\text{max}}^{\text{avg}}$ and is the validity domain $\mathbf{D} that is used to re-optimise the drift vector.

Fig. 8.4 illustrates a typical case where a single erroneous vector is surrounded by a smooth vector field. Since the central estimate is not used in the average, isolated wrong vectors stand out very easily in terms of their Δ_{avg}.

During this second CMCC optimization, the search for the maximum is limited to the area enclosed by the red circle. If a satisfying maximum correlation is found inside **D** it is kept and the surrounding average vectors are immediately updated, as well as each Δ_{avg} lengths. If the constrained optimization does not converge or if the new vector does not have a good enough maximum correlation value, both the old and new vectors are discarded and the average vectors, as well as Δ_{avg} at the neighboring locations are updated.

Although the method described above works in many cases, it sometimes fail when several erroneous vectors are close one to each other. This happens especially when noise dominates the signal in a large region of one of the image. If the case, the order in which the vectors are corrected has an influence on the final efficiency for the filtering.

To minimize this influence, motion vectors are first sorted from the largest to the shortest Δ_{avg} and the filtering is applied to the vector exhibiting the worst of those distances. Since, changing a vector has an influence on its direct neighbors, the sorting is repeated after each correction. A mechanism is put in place to avoid falling into an infinite loop. This strategy also ensures that the good vectors around an erroneous estimate are not modified before the latter is actually processed through the filter.

In the case where the new optimization does not lead to an acceptable maximum cross-correlation value (value below the threshold, non-convergence of the CMCC with the new contraint), the vector position is recorded as non-feasible in the status flags, and the fields of drift vectors get a fill value.

This detection / correction process continues until all vectors are either corrected or flagged as non-feasible.

Input data

The 2D field of drift vectors (δ_x and δ_y components) on the `(n,s)h_ease2-250` grid from the initial CMCC run.

The 2D field of maximum cross-correlation value for each vector from the initial CMCC run.

The 2D field of status flags recording the status at the end of the initial CMCC optimization.

Output data

Updates:

The 2D field of drift vectors (δ_x and δ_y components) on the `(n,s)h_ease2-250` grid from the initial CMCC run.

The 2D field of maximum cross-correlation value for each vector from the initial CMCC run.

The 2D field of status flags recording the status at the end of the initial CMCC optimization.

CHAPTER

NINE

ALGORITHM INPUT AND OUTPUT DATA DEFINITION (IODD)

Except for the input L1B TB data, the auxiliary and output data are all on *EASE2* grids. Refer to Table 8.1.

9.1 Input data

Field	Description	Shape/Amount
L1B TB	L1B Brightness Temperature at K and KA-bands (both H and V polarization)	full swath or section of it (Nscans, Npos)
L1B NeΔT	Random radiometric uncertainty of the channels	full swath or section of it (Nscans, Npos)

9.2 Output data

Field	Description	Shape/Amount
sea-ice drift vectors	dX and dY components of the motion vectors.	(nx,ny) on the `(n, s)h_ease2-250` grid.
sea-ice drift vectors uncertainty	dX and dY components of the uncertainty vector (1-sigma).	(nx,ny) on the `(n, s)h_ease2-250` grid.
status flags	indicate the reasons for missing, bad, or nominal vectors.	(nx,ny) on the `(n, s)h_ease2-250` grid.

9.3 Auxiliary data

Field	Description	Shape/Amount
sea-ice and land mask	A recent sea-ice concentration field including land information	SIC and land-mask on the `(n, s)h_ease2-005` grid.

CHAPTER

TEN

PREPROCESSING

CHAPTER

ELEVEN

LEVEL-2 SEA ICE DRIFT (SID) ALGORITHM FOR CIMR

This notebook implements a prototype for a Level-2 SIED algorithm for the CIMR mission.

We refer to the corresponding ATBD and especially the Baseline Algorithm Definition.

In particular, the figure below illustrates the overall concept of the processing:

11.1 Settings

Imports and general settings

```
# Paths

# Getting the path of the notebook (NOTE: not totally safe)

# The paths assume that there is an umbrella CIMR directory (of any name) containing␣
↪SeaIceDrift_ATBD_v2/ ,
# the CIMR Tools/ directory, and a directory data/L1B/ containing the L1B data, and␣
↪data/conc/ containing
# a concentration file
import os
cpath = os.path.join(os.getcwd(), '../..')
algpath = os.path.join(cpath, 'SeaIceDrift_ATBD_v2/algorithm/src_sied')
toolpath = os.path.join(cpath, 'Tools')
l1bpath = os.path.join(cpath, 'data/L1B')
griddeffile = os.path.join(cpath, 'Overall_ATBD/etc/grids_py.def')

# Creating a directory structure for processing
concpath = os.path.join(cpath, 'data/conc')
icemaskpath = os.path.join(cpath, 'data/icemask')
swathpath = os.path.join(cpath, 'data/swaths')
procpath = os.path.join(cpath, 'data/processing')
driftpath = os.path.join(cpath, 'data/icedrift')
logpath = os.path.join(cpath, 'data/logs')
figpath = os.path.join(cpath, 'data/figs')
for pth in [concpath, icemaskpath, swathpath, procpath, driftpath, logpath, figpath]:
    if not os.path.isdir(pth):
        os.makedirs(pth)
```

```
# Imports
from importlib import reload
import sys
```

(continues on next page)

(continued from previous page)

```
import warnings
from datetime import datetime, timedelta
from dateutil.relativedelta import relativedelta
import numpy as np
import xarray as xr
from netCDF4 import Dataset
from matplotlib import pylab as plt
import matplotlib.cm as cm
from pyresample import parse_area_file
from scipy.ndimage import laplace

# Local modules contain software code that implement the SIED algorithm
if algpath not in sys.path:
    sys.path.insert(0, algpath)
from process_ice_mask import process_ice_mask
from cp_and_date_change_iceconc import cp_and_date_change_iceconc

# Prototype re-gridding toolbox to handle the L1B input
if toolpath not in sys.path:
    sys.path.insert(0, toolpath)
from tools import io_handler as io
from tools import collocation as coll
from tools import l2_format as l2
```

```
# Plot settings

import matplotlib
matplotlib.rc('xtick', labelsize=10)
matplotlib.rc('ytick', labelsize=10)
matplotlib.rcParams.update({'font.size': 12})
matplotlib.rcParams.update({'axes.labelsize': 12})

font = {'family' : 'sans',
        'weight' : 'normal',
        'size'   : 12}
matplotlib.rc('font', **font)

cmap = cm.viridis
cmapland = matplotlib.colors.ListedColormap(['none', 'grey'])

gridin = '{}-ease2-050'
gridout = '{}-ease2-250'
# Some region parameters hard-coded to show only the relevant region
# Overall shape of input grid (4320, 4320)
sl = (1050, 1400, 1050, 1400)
```

11.2 Parametrize the run

User-set parameters for the running of the whole notebook. Note that here a helper script is used to copy a starter ice concentration file from the MET Norway thredds server and change the dates in this. The date changes are required due to the sample input file having a date in the future (2028).

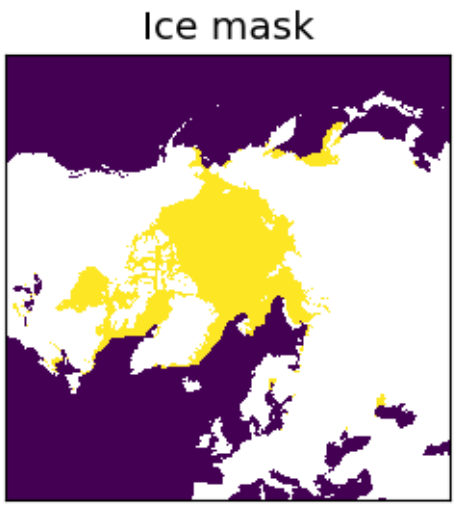
```
hemi = 'nh'
algos = {'KU': {'channels':('tb19v', 'tb19h'), 'target_band':'KU'},
         'KA': {'channels':('tb37v', 'tb37h'), 'target_band':'KA'}}
wbs = list(algos.keys())
fwdbck = ['fw', 'bk']
polarisation = {'V': 0, 'H': 1}
pols = list(polarisation.keys())

test_card = "radiometric"
if test_card == "geometric":
    # DEVALGO's simulated geometric test card
    l1bfn = 'W_PT-DME-Lisbon-SAT-CIMR-1B_C_DME_20230417T105425_LD_20280110T114800_
 ↪20280110T115700_TN.nc'
elif test_card == "radiometric":
    # DEVALGO's simulated radiometric test card
    l1bfn = 'W_PT-DME-Lisbon-SAT-CIMR-1B_C_DME_20230420T103323_LD_20280110T114800_
 ↪20280110T115700_TN.nc'

l1bfile = os.path.join(l1bpath, l1bfn)

pdate = datetime.strptime('20280110', '%Y%m%d')
tdate = pdate - relativedelta(years=10)
qdate = pdate + timedelta(days=1)

# Icemask data and output locations
icemaskinputdir = 'https://thredds.met.no/thredds/dodsC/osisaf/met.no/reprocessed/ice/
 ↪conc_450a_files/{:%Y}/{:%m}'.format(tdate, tdate)
icemaskinputfile = 'ice_conc_{}_ease2-250_cdr-v3p0_{:%Y%m%d}1200.nc'.format(hemi,␣
 ↪tdate)
icemaskinput = cp_and_date_change_iceconc(os.path.join(icemaskinputdir,␣
 ↪icemaskinputfile), concpath, pdate)

algo_version = '0.1'

pixshx = 3
pixshy = 4
```

```
Written a version of https://thredds.met.no/thredds/dodsC/osisaf/met.no/
 ↪reprocessed/ice/conc_450a_files/2018/01/ice_conc_nh_ease2-250_cdr-v3p0_
 ↪201801101200.nc to /home/emilyjd/cimr-devalgo/SeaIceDrift_ATBD_v2/book/../../
 ↪data/conc/ice_conc_nh_ease2-250_cdr-v3p0_202801101200.nc
```

11.3 Step 1: Creating and regridding the ice mask

A land/ocean/ice mask is required to define the areas with ice to the algorithm. This is created from a concentration file, and is stored for future use. Since there can be multiple ice drift calculations per day on different swaths, the mask can be reused once created.

```
# Creating the ice mask
gridname = gridin.format(hemi)
icemaskname = os.path.join(icemaskpath, 'icemask-multi-{}-{:%Y%m%d}12.nc'.
 ↪format(gridname, pdate))
if not os.path.isfile(icemaskname):
    # Suppress the proj4 string warning on this
    with warnings.catch_warnings():
        warnings.simplefilter("ignore")
        process_ice_mask(icemaskinput, icemaskpath, griddeffile, gridname)

# Reading in the ice mask
ie_data = Dataset(icemaskname, 'r')
ie = ie_data['ice_edge'][0, :, :]

# And the same for the output grid
gridnameout = gridout.format(hemi)
icemasknameout = os.path.join(icemaskpath, 'icemask-multi-{}-{:%Y%m%d}12.nc'.
 ↪format(gridnameout, pdate))
if not os.path.isfile(icemasknameout):
    # Suppress the proj4 string warning on this
    with warnings.catch_warnings():
        warnings.simplefilter("ignore")
        process_ice_mask(icemaskinput, icemaskpath, griddeffile, gridnameout)
ie_data_out = Dataset(icemasknameout, 'r')
ieout = ie_data_out['ice_edge'][0, :, :]

# Plotting the ice mask
fig = plt.figure(figsize=(3,3))
ax1 = fig.add_subplot(1,1,1)
c1 = ax1.imshow(ie[:], interpolation = 'none', cmap=cmap)
ax1.set_title("Ice mask")
ax1.xaxis.set_tick_params(labelbottom=False)
ax1.yaxis.set_tick_params(labelleft=False)
ax1.set_xticks([])
ax1.set_yticks([])

# Input landmask
landmask = np.zeros_like(ie)
landmask[ie == 9] = 1

# Output landmask
landmaskout = np.zeros_like(ieout)
landmaskout[ieout == 9] = 1
```



11.4 Step 2: Loading the data

The L1B data is read in and split into forward and backward scans using software from the `Tools/` repository (a prototype CIMR Regridding Toolbox developed in the CIMR DEVALGO study). These forward and backward scans can be used independently in the algorithm in the same way as different channels and polarisations.

```
# Read the bands needed of the L1B data
reload(io)

tb_dict = {'tb01':'L', 'tb06':'C', 'tb10':'X', 'tb19':'KU', 'tb37':'KA'}
rev_tb_dict = {v:k for k,v in tb_dict.items()}
bands_needed = []
for alg in algos.keys():
    bands_needed += algos[alg]['channels']
bands_needed = list(set([tb_dict[b[:-1]] for b in bands_needed]))

full_l1b = io.CIMR_L1B(l1bfile, selected_bands=bands_needed, keep_calibration_
 ↪view=True)
```

```
# Split into forward / backward scan

l1b = {}
l1b['fw'], l1b['bk'] = full_l1b.split_forward_backward_scans(method='horn_scan_angle')
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[7], line 4
      1 # Split into forward / backward scan
      3 l1b = {}
----> 4 l1b['fw'], l1b['bk'] = full_l1b.split_forward_backward_scans(method='horn_
 ↪scan_angle')

File ~/cimr-devalgo/SeaIceDrift_ATBD_v2/book/../../Tools/tools/io_handler.py:328,␣
 ↪in CIMR_L1B.split_forward_backward_scans(self, method)
    325     bck_mask = (horn_scan_angle >= 180.)*(horn_scan_angle < 360.)
```

(continues on next page)

(continued from previous page)

```
    327 # do the selection
--> 328 fwd_l1b.data[band] = self.data[band].where(fwd_mask, drop=True)
    329 bck_l1b.data[band] = self.data[band].where(bck_mask, drop=True)
    331 # ensure that the first three dimensions stay ('n_scans', 'n_samples_earth
 ↪', 'n_horns')
    332 #   sometimes xarray creates a n_horns dimension at the end of the␣
 ↪dimension tuplet, but
    333 #   most of our software relies on having these three dims first

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/xarray/core/
 ↪common.py:1246, in DataWithCoords.where(self, cond, other, drop)
   1243     self = self.isel(**indexers)
   1244     cond = cond.isel(**indexers)
-> 1246 return ops.where_method(self, cond, other)

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/xarray/core/
 ↪ops.py:179, in where_method(self, cond, other)
    177 # alignment for three arguments is complicated, so don't support it yet
    178 join = "inner" if other is dtypes.NA else "exact"
--> 179 return apply_ufunc(
    180     duck_array_ops.where_method,
    181     self,
    182     cond,
    183     other,
    184     join=join,
    185     dataset_join=join,
    186     dask="allowed",
    187     keep_attrs=True,
    188 )

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/xarray/core/
 ↪computation.py:1253, in apply_ufunc(func, input_core_dims, output_core_dims,␣
 ↪exclude_dims, vectorize, join, dataset_join, dataset_fill_value, keep_attrs,␣
 ↪kwargs, dask, output_dtypes, output_sizes, meta, dask_gufunc_kwargs, on_missing_
 ↪core_dim, *args)
   1251 # feed datasets apply_variable_ufunc through apply_dataset_vfunc
   1252 elif any(is_dict_like(a) for a in args):
-> 1253     return apply_dataset_vfunc(
   1254         variables_vfunc,
   1255         *args,
   1256         signature=signature,
   1257         join=join,
   1258         exclude_dims=exclude_dims,
   1259         dataset_join=dataset_join,
   1260         fill_value=dataset_fill_value,
   1261         keep_attrs=keep_attrs,
   1262         on_missing_core_dim=on_missing_core_dim,
   1263     )
   1264 # feed DataArray apply_variable_ufunc through apply_dataarray_vfunc
   1265 elif any(isinstance(a, DataArray) for a in args):

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/xarray/core/
 ↪computation.py:528, in apply_dataset_vfunc(func, signature, join, dataset_join,␣
 ↪fill_value, exclude_dims, keep_attrs, on_missing_core_dim, *args)
    523 list_of_coords, list_of_indexes = build_output_coords_and_indexes(
    524     args, signature, exclude_dims, combine_attrs=keep_attrs
```

(continues on next page)

(continued from previous page)

```
    525 )
    526 args = tuple(getattr(arg, "data_vars", arg) for arg in args)
--> 528 result_vars = apply_dict_of_variables_vfunc(
    529     func,
    530     *args,
    531     signature=signature,
    532     join=dataset_join,
    533     fill_value=fill_value,
    534     on_missing_core_dim=on_missing_core_dim,
    535 )
    537 out: Dataset | tuple[Dataset, ...]
    538 if signature.num_outputs > 1:

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/xarray/core/
↪computation.py:452, in apply_dict_of_variables_vfunc(func, signature, join, fill_
↪value, on_missing_core_dim, *args)
    450 core_dim_present = _check_core_dims(signature, variable_args, name)
    451 if core_dim_present is True:
--> 452     result_vars[name] = func(*variable_args)
    453 else:
    454     if on_missing_core_dim == "raise":

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/xarray/core/
↪computation.py:730, in apply_variable_ufunc(func, signature, exclude_dims, dask,␣
↪output_dtypes, vectorize, keep_attrs, dask_gufunc_kwargs, *args)
    725 broadcast_dims = tuple(
    726     dim for dim in dim_sizes if dim not in signature.all_core_dims
    727 )
    728 output_dims = [broadcast_dims + out for out in signature.output_core_dims]
--> 730 input_data = [
    731     (
    732         broadcast_compat_data(arg, broadcast_dims, core_dims)
    733         if isinstance(arg, Variable)
    734         else arg
    735     )
    736     for arg, core_dims in zip(args, signature.input_core_dims)
    737 ]
    739 if any(is_chunked_array(array) for array in input_data):
    740     if dask == "forbidden":

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/xarray/core/
↪computation.py:732, in <listcomp>(.0)
    725 broadcast_dims = tuple(
    726     dim for dim in dim_sizes if dim not in signature.all_core_dims
    727 )
    728 output_dims = [broadcast_dims + out for out in signature.output_core_dims]
    730 input_data = [
    731     (
--> 732         broadcast_compat_data(arg, broadcast_dims, core_dims)
    733         if isinstance(arg, Variable)
    734         else arg
    735     )
    736     for arg, core_dims in zip(args, signature.input_core_dims)
    737 ]
    739 if any(is_chunked_array(array) for array in input_data):
    740     if dask == "forbidden":
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/xarray/core/
↪computation.py:653, in broadcast_compat_data(variable, broadcast_dims, core_dims)
    648 def broadcast_compat_data(
    649     variable: Variable,
    650     broadcast_dims: tuple[Hashable, ...],
    651     core_dims: tuple[Hashable, ...],
    652 ) -> Any:
--> 653     data = variable.data
    655     old_dims = variable.dims
    656     new_dims = broadcast_dims + core_dims

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/xarray/core/
↪variable.py:448, in Variable.data(self)
    446     return self._data
    447 elif isinstance(self._data, indexing.ExplicitlyIndexed):
--> 448     return self._data.get_duck_array()
    449 else:
    450     return self.values

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/xarray/core/
↪indexing.py:836, in MemoryCachedArray.get_duck_array(self)
    835 def get_duck_array(self):
--> 836     self._ensure_cached()
    837     return self.array.get_duck_array()

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/xarray/core/
↪indexing.py:830, in MemoryCachedArray._ensure_cached(self)
    829 def _ensure_cached(self):
--> 830     self.array = as_indexable(self.array.get_duck_array())

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/xarray/core/
↪indexing.py:787, in CopyOnWriteArray.get_duck_array(self)
    786 def get_duck_array(self):
--> 787     return self.array.get_duck_array()

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/xarray/core/
↪indexing.py:650, in LazilyIndexedArray.get_duck_array(self)
    646     array = apply_indexer(self.array, self.key)
    647 else:
    648     # If the array is not an ExplicitlyIndexedNDArrayMixin,
    649     # it may wrap a BackendArray so use its __getitem__
--> 650     array = self.array[self.key]
    652 # self.array[self.key] is now a numpy array when
    653 # self.array is a BackendArray subclass
    654 # and self.key is BasicIndexer((slice(None, None, None),))
    655 # so we need the explicit check for ExplicitlyIndexed
    656 if isinstance(array, ExplicitlyIndexed):

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/xarray/
↪backends/netCDF4_.py:100, in NetCDF4ArrayWrapper.__getitem__(self, key)
     99 def __getitem__(self, key):
--> 100     return indexing.explicit_indexing_adapter(
    101         key, self.shape, indexing.IndexingSupport.OUTER, self._getitem
    102     )
```

(continues on next page)

(continued from previous page)

```
File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/xarray/core/
 ↪indexing.py:1014, in explicit_indexing_adapter(key, shape, indexing_support, raw_
 ↪indexing_method)
    992 """Support explicit indexing by delegating to a raw indexing method.
    993
    994 Outer and/or vectorized indexers are supported by indexing a second time
   (...)
   1011 Indexing result, in the form of a duck numpy-array.
   1012 """
   1013 raw_key, numpy_indices = decompose_indexer(key, shape, indexing_support)
-> 1014 result = raw_indexing_method(raw_key.tuple)
   1015 if numpy_indices.tuple:
   1016     # index the loaded np.ndarray
   1017     indexable = NumpyIndexingAdapter(result)

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/xarray/
 ↪backends/netCDF4_.py:113, in NetCDF4ArrayWrapper._getitem(self, key)
    111     with self.datastore.lock:
    112         original_array = self.get_array(needs_lock=False)
--> 113         array = getitem(original_array, key)
    114 except IndexError:
    115     # Catch IndexError in netCDF4 and return a more informative
    116     # error message.  This is most often called when an unsorted
    117     # indexer is used before the data is loaded from disk.
    118     msg = (
    119         "The indexing operation you are attempting to perform "
    120         "is not valid on netCDF4.Variable object. Try loading "
    121         "your data into memory first by calling .load()."
    122     )

KeyboardInterrupt:
```

11.5 Step 3: Regridding the data

The horns are interleaved, and then the data is regridded. These are again done with software from `Tools/`. The ice drift will be calculated individually on 8 fields based on the forward and backward scans, waveband (Ku or Ka), and polarity (V or H). A fine EASE2 grid spacing of 5km is chosen for this regridding.

```
# Regridding the data

reload(coll)

# Reshaping
l1b_r = {}
for fb in fwdbck:
    l1b_r[fb] = l1b[fb].reshape_interleave_feed()

# Loading the target grid information
gridname = gridin.format(hemi)
new_area_def = parse_area_file(griddeffile, gridname)[0]
new_lons, new_lats = new_area_def.get_lonlats()

# Getting the input lat/lons
```

(continues on next page)

(continued from previous page)

```
lonlats = {}
for ll in ['lon', 'lat']:
    lonlats[ll] = {}
    for fb in fwdbck:
        lonlats[ll][fb] = {}
        for wb in wbs:
            lonlats[ll][fb][wb] = l1b_r[fb].data[wb][ll].data

# Creating data arrays with the V and H layers
what = ('brightness_temperature_v', 'brightness_temperature_h')
params = {'method':'gauss', 'sigmas':25000, 'neighbours':55}
stack_shape = {}
stack = {}
regrid = {}
for fb in fwdbck:
    stack_shape[fb] = {}
    stack[fb] = {}
    regrid[fb] = {}
    for wb in wbs:
        stack_shape[fb][wb] = tuple(list(lonlats['lat'][fb][wb].shape) + [len(what),])
        stack[fb][wb] = np.empty(stack_shape[fb][wb])
        for iw, w in enumerate(what):
            stack[fb][wb][...,iw] = l1b_r[fb].data[wb][w].data
            # Regridding
            # TODO - Add passing of params, currently a nearest-neighbour approach␣
 ↪with ROI 15000 is used
            regrid[fb][wb] = coll._regrid_fields(new_lons, new_lats,
                                                 lonlats['lon'][fb][wb], lonlats['lat
 ↪'][fb][wb], stack[fb][wb])
```

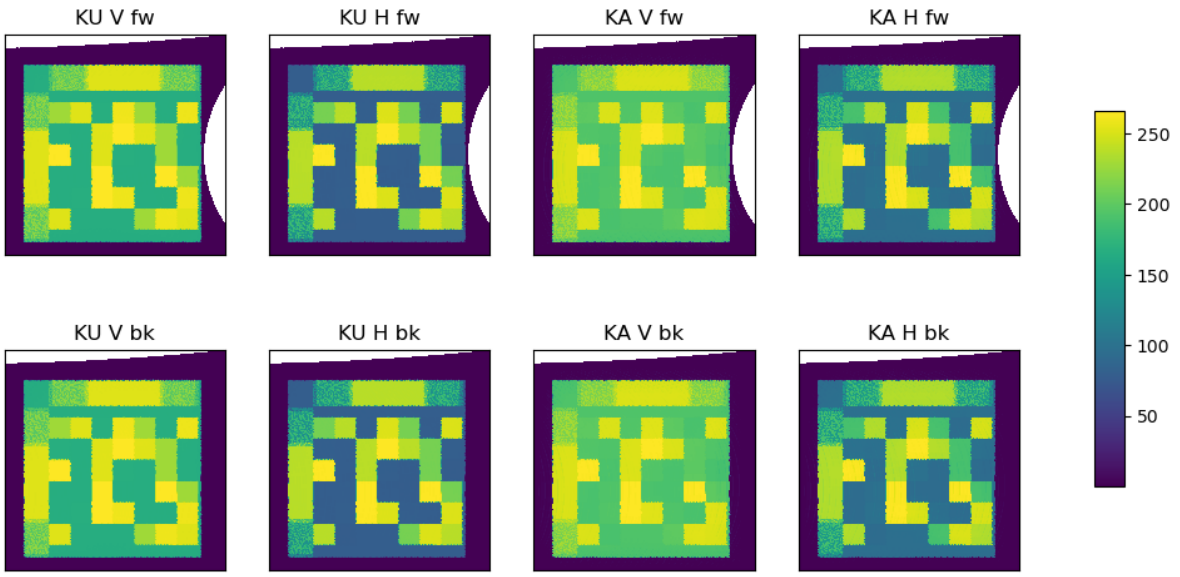
```
# Plot regridded

fig = plt.figure(figsize=(12,6))
ax = {}
c = {}
shapelayout = (len(fwdbck), len(wbs) * len(pols))
axindex = 1
for fb in fwdbck:
    for wb in wbs:
        for pol in pols:
            ax[axindex] = fig.add_subplot(*shapelayout, axindex)
            c[axindex] = ax[axindex].imshow(regrid[fb][wb][:, :,␣
 ↪polarisation[pol]][sl[0]:sl[1], sl[2]:sl[3]],
                                            interpolation = 'none', origin='lower',␣
 ↪cmap=cmap)
            overland = ax[axindex].imshow(landmask[sl[0]:sl[1], sl[2]:sl[3]],␣
 ↪interpolation='none',
                                          origin='lower', cmap=cmapland)
            ax[axindex].invert_yaxis()
            ax[axindex].set_title("{} {} {}".format(wb, pol, fb), fontsize=12)
            ax[axindex].xaxis.set_tick_params(labelbottom=False)
            ax[axindex].yaxis.set_tick_params(labelleft=False)
            ax[axindex].set_xticks([])
            ax[axindex].set_yticks([])
            axindex += 1
```

(continues on next page)

(continued from previous page)

```
fig.subplots_adjust(right=0.8)
cbar_ax = fig.add_axes([0.85, 0.25, 0.02, 0.5])
fig.colorbar(c[1], cax=cbar_ax, shrink=0.5)
```

```
<matplotlib.colorbar.Colorbar at 0x7fc98da9a190>
```



11.6 Step 4: Laplacian pre-processing

Instead of directly using the brightness temperatures in the motion tracking algorithm, the ice features are stabilised and enhanced by applying a Laplacian filter (see ATBD for mathmatical description).

```
# Replace fill value by NaN and remove mask
def _get_nans(img):
    img_masked = np.ma.asarray(img)
    return img_masked.filled(np.nan)

# Replace NaN by fill value and add mask
def _mask_nans(img):
    return np.ma.masked_invalid(img)

nan = {}
lap = {}
fv = {}
for fb in fwdbck:
    nan[fb] = {}
    lap[fb] = {}
    fv[fb] = {}
    for wb in wbs:
        # Convert fill values to NaNs
        nan[fb][wb] = _get_nans(regrid[fb][wb])
        # Laplacian transform
        lap[fb][wb] = laplace(nan[fb][wb])
```

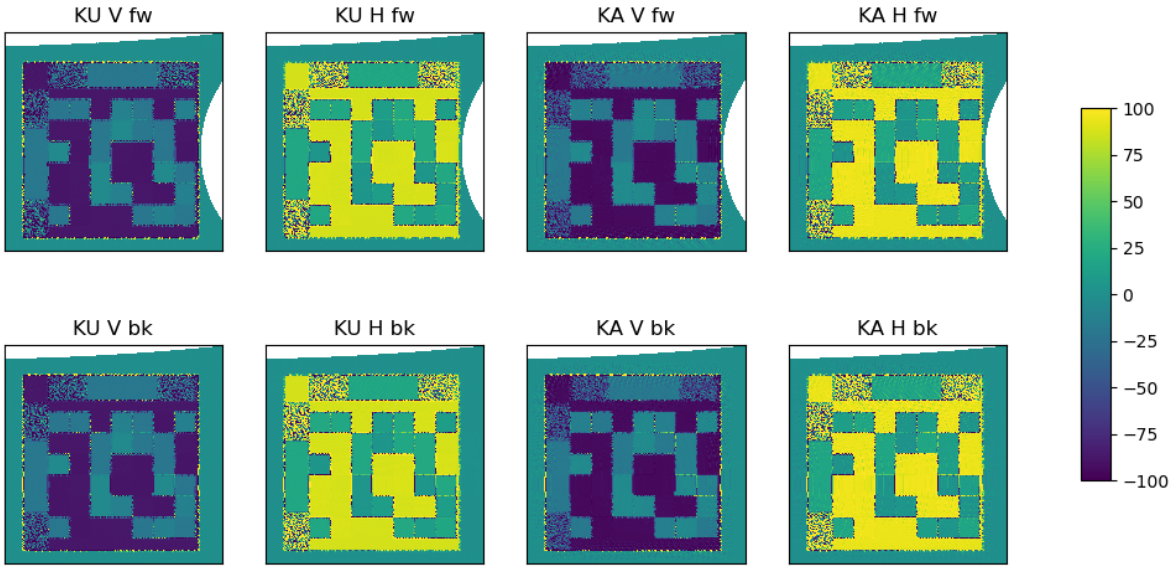
(continues on next page)

(continued from previous page)

```
        # Converting NaNs to fill values
        fv[fb][wb] = _mask_nans(lap[fb][wb])

# Creating a flag field
#define TCIMAGE_OUTSIDE_GRID              -2
#define TCIMAGE_NODATA                    -1
#define TCIMAGE_OK                         0
#define TCIMAGE_UNPROCESSED                1
#define TCIMAGE_FAILED                     2
flag = {}
for fb in fwdbck:
    flag[fb] = {}
    for wb in wbs:
        flag[fb][wb] = np.zeros_like(fv[fb][wb])
        # Masking where the Laplacian didn't work
        flag[fb][wb][fv[fb][wb].mask] = -1
        # Masking where the land and ocean is
        landocean = np.logical_or(ie == 9, ie == 1)
        flag[fb][wb][landocean] = 1
```

```
# Plot Laplacian

vmin = -100
vmax = 100

fig = plt.figure(figsize=(12,6))
ax = {}
c = {}
shapelayout = (len(fwdbck), len(wbs) * len(pols))
axindex = 1
for fb in fwdbck:
    for wb in wbs:
        for pol in pols:
            ax[axindex] = fig.add_subplot(*shapelayout, axindex)
            c[axindex] = ax[axindex].imshow(fv[fb][wb][:, :,␣
 ↪polarisation[pol]][sl[0]:sl[1], sl[2]:sl[3]],
                                            interpolation = 'none', origin='lower',␣
 ↪cmap=cmap,
                                            vmin=vmin, vmax=vmax)
            overland = ax[axindex].imshow(landmask[sl[0]:sl[1], sl[2]:sl[3]],␣
 ↪interpolation='none',
                                          origin='lower', cmap=cmapland)
            ax[axindex].invert_yaxis()
            ax[axindex].set_title("{} {} {}".format(wb, pol, fb), fontsize=12)
            ax[axindex].xaxis.set_tick_params(labelbottom=False)
            ax[axindex].yaxis.set_tick_params(labelleft=False)
            ax[axindex].set_xticks([])
            ax[axindex].set_yticks([])
            axindex += 1

fig.subplots_adjust(right=0.8)
cbar_ax = fig.add_axes([0.85, 0.25, 0.02, 0.5])
fig.colorbar(c[1], cax=cbar_ax, shrink=0.5)
```

```
<matplotlib.colorbar.Colorbar at 0x7fc98787f4f0>
```



11.7 Step 5: Writing out the file

For ice drift, it is not possible to keep the data only internally. Since two gridded files at different timepoints are required to create each gridded map of icedrift vectors, it is necessary to write each gridded swath file to disk so that it can be read in to be compared with other gridded swath files.

```
# Note: It is deprecated, but the wrapper and C code still expect a input proj4_
 ↪string, so accept the
# deprecation warning for now.
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    crs_info = {'proj4_string': new_area_def.proj4_string,
                'area_id': new_area_def.area_id,
                'semi_major_axis': 6378137.,
                'semi_minor_axis': 6356752.31424518,
                'inverse_flattening': 298.257223563,
                'reference_ellipsoid_name': "WGS 84",
                'longitude_of_prime_meridian': 0.,
                'prime_meridian_name': "Greenwich",
                'geographic_crs_name': "unknown",
                'horizontal_datum_name': "World Geodetic System 1984",
                'projected_crs_name': "unknown",
                'grid_mapping_name': "lambert_azimuthal_equal_area",
                'latitude_of_projection_origin': 90.,
                'longitude_of_projection_origin': 0.,
                'false_easting': 0.,
                'false_northing': 0.}
```

```
reload(l2)

# Get a template L2 format (netCDF/CF) from the Tools module
ds_l2 = l2.get_CIMR_L2_template('grid', geo_def=new_area_def, add_time=[pdate.
 ↪timestamp()])
```

(continues on next page)

(continued from previous page)

```
# Create data arrays for the brightness temperatures, laplacian processed and status␣
 ↪flags from the template
shp = (1, *regrid[fwdbck[0]][wbs[0]][:, :, polarisation[pol]].shape)
ds_tb = {}
ds_lap = {}
ds_flag = {}
for fb in fwdbck:
    ds_tb[fb] = {}
    ds_lap[fb] = {}
    ds_flag[fb] = {}
    for wb in wbs:
        ds_tb[fb][wb] = {}
        ds_lap[fb][wb] = {}
        ds_flag[fb][wb] = {}
        for pol in pols:
            chan = algos[wb]['channels'][polarisation[pol]]
            ds_tb[fb][wb][pol] = xr.DataArray(regrid[fb][wb][:, :, polarisation[pol]].
 ↪reshape(shp),
                                              coords=ds_l2['template'].coords,␣
 ↪dims=ds_l2['template'].dims,
                                              attrs=ds_l2['template'].attrs, name='{}
 ↪{}'.format(chan, fb))
            ds_tb[fb][wb][pol].attrs['standard_name'] = 'brightness_temperature'
            ds_tb[fb][wb][pol].attrs['long_name'] = 'Brightness temperature {}'.
 ↪format(chan, fb)
            ds_tb[fb][wb][pol].attrs['coverage_content_type'] = 'physicalMeasurement'
            ds_tb[fb][wb][pol].attrs['units'] = 'K'
            ds_l2 = ds_l2.merge(ds_tb[fb][wb][pol])

            ds_lap[fb][wb][pol] = xr.DataArray(fv[fb][wb][:, :, polarisation[pol]].
 ↪reshape(shp),
                                               coords=ds_l2['template'].coords,␣
 ↪dims=ds_l2['template'].dims,
                                               attrs=ds_l2['template'].attrs, name='{}
 ↪{}_lap'.format(chan, fb))
            ds_lap[fb][wb][pol].attrs['long_name'] = 'Laplacian of brightness␣
 ↪temperature {}'.format(chan, fb)
            ds_lap[fb][wb][pol].attrs['coverage_content_type'] = 'auxiliaryInformation
 ↪'
            ds_lap[fb][wb][pol].attrs['units'] = 1
            ds_l2 = ds_l2.merge(ds_lap[fb][wb][pol])

            ds_flag[fb][wb][pol] = xr.DataArray(flag[fb][wb][:, :, polarisation[pol]].
 ↪reshape(shp),
                                                coords=ds_l2['template'].coords,␣
 ↪dims=ds_l2['template'].dims,
                                                attrs=ds_l2['template'].attrs,
                                                name='{}{}_lap_flag'.format(chan, fb))
            ds_flag[fb][wb][pol].attrs['standard_name'] = 'status_flag'
            ds_flag[fb][wb][pol].attrs['long_name'] = 'Status flag for Laplacian of␣
 ↪brightness temperature {}'.format(chan, fb)
            ds_flag[fb][wb][pol].attrs['coverage_content_type'] = 'qualityInformation'
            ds_flag[fb][wb][pol].attrs['units'] = 1
            ds_l2 = ds_l2.merge(ds_flag[fb][wb][pol])
```

(continues on next page)

(continued from previous page)

```
# Create a data array for dtime from the template
dtime = np.full_like(regrid[fwdbck[0]][wbs[0]][:, :, 0], pdate.timestamp()).
 ↪reshape(shp)
ds_dtime = xr.DataArray(dtime, coords=ds_l2['template'].coords, dims=ds_l2['template
 ↪'].dims,
                        attrs=ds_l2['template'].attrs, name='dtime')
ds_dtime.attrs['long_name'] = 'Time'
ds_dtime.attrs['standard_name'] = 'time'
ds_dtime.attrs['coverage_content_type'] = 'auxiliaryInformation'
ds_dtime.attrs['units'] = 'seconds since 1970-01-01 00:00:00'
ds_l2 = ds_l2.merge(ds_dtime)

# Create a data array for ice edge from the template
ds_ie = xr.DataArray(ie.reshape(shp), coords=ds_l2['template'].coords, dims=ds_l2[
 ↪'template'].dims,
                     attrs=ds_l2['template'].attrs, name='ice_edge')
ds_ie.attrs['long_name'] = 'Ice edge'
ds_ie.attrs['coverage_content_type'] = 'auxiliaryInformation'
ds_ie.attrs['units'] = 1
ds_l2 = ds_l2.merge(ds_ie)

# Create a data array for time, needed by the C code
timedata = np.full_like(regrid[fb][wb][0, 0, 0], pdate.timestamp())
ds_l2['vtime'] = (('time'), timedata.reshape(1,))
ds_l2['time'].attrs = {'units': "seconds since 1970-01-01 00:00:00"}
ds_l2.attrs['long_name'] = 'Time'
ds_l2.attrs['coverage_content_type'] = 'auxiliaryInformation'
ds_l2.attrs['units'] = "seconds since 1970-01-01 00:00:00"

# Customize the global attributes
ds_l2.attrs['title'] = 'CIMR intermediate brightness temperatures for ice drift␣
 ↪calculation'
ds_l2.attrs['summary'] = 'Intermediate brightness temperatures and Laplacian-
 ↪processed fields with their status flags, written as intermadiate-processing file␣
 ↪for ice drift calculations'
ds_l2.attrs['l1b_file'] = l1bfn
ds_l2.attrs['algorithm_version'] = algo_version
ds_l2.attrs['creator_name'] = 'Emily Down'
ds_l2.attrs['creator_email'] = 'emilyjd@met.no'
ds_l2.attrs['institution'] = 'Norwegian Meteorological Institute'

# CRS information needed for C code
ds_l2['crs'].attrs = crs_info

# Need to rename x and y to xc and yc for X code
ds_l2= ds_l2.rename({'x': 'xc','y': 'yc'})

# Remove the 'template' variable (we don't need it anymore)
#ds_l2 = ds_l2.drop('template')

# Write to file
dsname = os.path.join(procpath, 'bt_{}_{:%Y%m%d}.nc'.format(gridname, pdate))
ds_l2.to_netcdf(dsname, 'w', format="NETCDF4_CLASSIC")
print("Written {}".format(dsname))

# Setting this up for potential reruns
ds_l2_copy = ds_l2.copy()
```

```
Written /home/emilyjd/cimr-devalgo/SeaIceDrift_ATBD_v2/algorithm/../../data/
↪processing/bt_nh-ease2-050_20280110.nc
```

11.8 Step 6 (temporary): Creating a test gridded file with "time difference" 24h and pixel shifts added in each x and y

This is a temporary part of this notebook, to create a "test" file with different gridded brightness temperatures and a different timepoint. This is required for calculation of ice drift vectors. In this example, a constant shift of +3 pixels in the x-direction and +4 pixels in the y-direction is chosen. In operation, two swaths at different timepoints will be processed in the algorithm to retrieve the icedrift.

```
# Create a shifted file with fixed pixels.

# Take a copy of the data xarray
ds_shift = ds_l2_copy.copy()

# Add the Laplacian mask variable
ds_msk = {}
ds_msk = {}
ds_msk = {}
for fb in fwdbck:
    ds_msk[fb] = {}
    for wb in wbs:
        ds_msk[fb][wb] = {}
        for pol in pols:
            chan = algos[wb]['channels'][polarisation[pol]]
            ds_msk[fb][wb][pol] = xr.DataArray(fv[fb][wb][:, :, polarisation[pol]].
↪mask.reshape(shp),
                                               coords=ds_shift['template'].coords,␣
↪dims=ds_shift['template'].dims,
                                               attrs=ds_shift['template'].attrs, name=
↪'{}{}_msk'.format(chan, fb))
            ds_msk[fb][wb][pol].attrs['long_name'] = 'Mask for Laplacian of␣
↪brightness temperature {}'.format(chan, fb)
            ds_msk[fb][wb][pol].attrs['coverage_content_type'] = 'auxiliaryInformation
↪'
            ds_msk[fb][wb][pol].attrs['units'] = 1
            ds_shift = ds_shift.merge(ds_msk[fb][wb][pol])

# Shifting the data pixels of the arrays
ds_shift_tb = {}
ds_shift_lap = {}
ds_shift_flag = {}
for fb in fwdbck:
    ds_shift_tb[fb] = {}
    ds_shift_lap[fb] = {}
    ds_shift_flag[fb] = {}
    for wb in wbs:
        ds_shift_tb[fb][wb] = {}
        ds_shift_lap[fb][wb] = {}
        ds_shift_flag[fb][wb] = {}
        for pol in pols:
            chan = algos[wb]['channels'][polarisation[pol]]
```

(continues on next page)

(continued from previous page)

```
            # Shift can be used in xarray to shift dimension by number of pix. Note␣
 ↪that the dimensions must be
            # called x and y within xarray
            chan = algos[wb]['channels'][polarisation[pol]]
            tbname = '{}{}'.format(chan, fb)
            ds_shift_tb[fb][wb] = ds_shift.get(tbname).shift(xc=pixshx, yc=pixshy)
            ds_shift[tbname].data = ds_shift_tb[fb][wb]
            # Shifting the Laplacian field
            lapname = '{}{}_lap'.format(chan, fb)
            ds_shift_lap[fb][wb] = ds_shift.get(lapname).shift(xc=pixshx, yc=pixshy)
            ds_shift[lapname].data = ds_shift_lap[fb][wb]
            # Flag field (want the data shifted, the landmask not)
            flagname = '{}{}_lap_flag'.format(chan, fb)
            ds_shift_flag[fb][wb] = np.zeros_like(ds_shift_lap[fb][wb])
            # Masking where the Laplacian failed
            maskname = '{}{}_msk'.format(chan, fb)
            fmsk_shft = ds_shift.get(maskname).shift(xc=pixshx, yc=pixshy)
            ds_shift_flag[fb][wb][fmsk_shft == 1] = -1
            # Masking where the land and ocean is
            landocean = np.logical_or(ie == 9, ie == 1).reshape(shp)
            (ds_shift_flag[fb][wb])[landocean] = 1
            ds_shift[flagname].data = ds_shift_flag[fb][wb]

# Shift time by 24h
ds_shift['dtime'].data = np.full_like(regrid[fwdbck[0]][wbs[0]][:, :, 0], qdate.
 ↪timestamp()).reshape(shp)
ds_shift.assign_coords(time = [qdate.timestamp()])

# Remove the mask variables (we don't need them anymore)
for fb in fwdbck:
    for wb in wbs:
        for pol in pols:
            chan = algos[wb]['channels'][polarisation[pol]]
            maskname = '{}{}_msk'.format(chan, fb)
            ds_shift = ds_shift.drop_vars(maskname)

# Write to file
dsname2 = os.path.join(procpath, 'bt_{}_{:%Y%m%d}.nc'.format(gridname, qdate))
ds_shift.to_netcdf(dsname2, 'w', format="NETCDF4_CLASSIC")
print("Written {}".format(dsname2))
```

```
Written /home/emilyjd/cimr-devalgo/SeaIceDrift_ATBD_v2/algorithm/../../data/
 ↪processing/bt_nh-ease2-050_20280111.nc
```

CHAPTER

TWELVE

ALGORITHM

CHAPTER

THIRTEEN

LEVEL-2 SEA ICE DRIFT (SID) ALGORITHM FOR CIMR

This notebook implements a prototype for a Level-2 SIED algorithm for the CIMR mission.

We refer to the corresponding ATBD and especially the Baseline Algorithm Definition.

In particular, the figure below illustrates the overall concept of the processing:

13.1 Settings

Imports and general settings

```
%load_ext cython
```

```
# Paths

# Getting the path of the notebook (NOTE: not totally safe)

# The paths assume that there is an umbrella CIMR directory (of any name) containing␣
 ↪SeaIceDrift_ATBD_v2/ ,
# the CIMR Tools/ directory, and a directory data/L1B/ containing the L1B data, and␣
 ↪data/conc/ containing
# a concentration file
import os
cpath = os.path.join(os.getcwd(), '../..')
algpath = os.path.join(cpath, 'SeaIceDrift_ATBD_v2/algorithm/src_sied')
toolpath = os.path.join(cpath, 'Tools')
l1bpath = os.path.join(cpath, 'data/L1B')
griddeffile = os.path.join(cpath, 'Overall_ATBD/etc/grids_py.def')

# Processing directories
procpath = os.path.join(cpath, 'data/processing')
driftpath = os.path.join(cpath, 'data/icedrift')
logpath = os.path.join(cpath, 'data/logs')
figpath = os.path.join(cpath, 'data/figs')
```

```
# Imports

from importlib import reload
import sys
import shutil
import warnings
```

(continues on next page)

(continued from previous page)

```
import math
import numpy as np
import numpy.ma as ma
import xarray as xr
#from netCDF4 import Dataset
from matplotlib import pylab as plt
import matplotlib.ticker as mticker
from mpl_toolkits.axes_grid1.axes_divider import make_axes_locatable
import matplotlib.cm as cm
#import cmocean
#import cartopy
import cartopy.crs as ccrs
from pyresample import parse_area_file
from datetime import datetime, timedelta
from dateutil.relativedelta import relativedelta

# Local modules contain software code that implement the SIED algorithm
if algpath not in sys.path:
    sys.path.insert(0, algpath)
from icedrift_wrapper import icedrift_wrapper
#from process_ice_mask import process_ice_mask
#from cp_and_date_change_iceconc import cp_and_date_change_iceconc

# Prototype re-gridding toolbox to handle the L1B input
if toolpath not in sys.path:
    sys.path.insert(0, toolpath)
#from tools import io_handler as io
#from tools import collocation as coll
from tools import l2_format as l2
```

```
# Plot settings

import matplotlib
matplotlib.rc('xtick', labelsize=10)
matplotlib.rc('ytick', labelsize=10)
matplotlib.rcParams.update({'font.size': 12})
matplotlib.rcParams.update({'axes.labelsize': 12})

font = {'family' : 'sans',
        'weight' : 'normal',
        'size'   : 12}
matplotlib.rc('font', **font)

cmap = cm.viridis
#cmapland = matplotlib.colors.ListedColormap(['none', 'grey'])

gridtype = 'ease'
gridin = '{}-ease2-050'
gridout = '{}-ease2-250'
# Some region parameters hard-coded to show only the relevant region
# Overall shape of input grid (4320, 4320)
#sl = (1050, 1400, 1050, 1400)
slo = (200, 290, 200, 290)

# EASE plotting region
lon_min = -15
```

(continues on next page)

(continued from previous page)

```
lon_max = 95
lat_min = 74
lat_max = 90

# Settings for gridlines
lon_step = 10
lat_step = 5
```

13.2 Parametrize the run

User-set parameters for the running of the whole notebook. Note that here a helper script is used to copy a starter ice concentration file from the MET Norway thredds server and change the dates in this. The date changes are required due to the sample input file having a date in the future (2028).

```
hemi = 'nh'
#algos = {'KU': {'channels':('tb19v', 'tb19h'), 'target_band':'KU'},
#         'KA': {'channels':('tb37v', 'tb37h'), 'target_band':'KA'}}
#wbs = list(algos.keys())
#fwdbck = ['fw', 'bk']
#polarisation = {'V': 0, 'H': 1}
#pols = list(polarisation.keys())

test_card = "radiometric"
if test_card == "geometric":
    # DEVALGO's simulated geometric test card
    l1bfn = 'W_PT-DME-Lisbon-SAT-CIMR-1B_C_DME_20230417T105425_LD_20280110T114800_
 ↪20280110T115700_TN.nc'
elif test_card == "radiometric":
    # DEVALGO's simulated radiometric test card
    l1bfn = 'W_PT-DME-Lisbon-SAT-CIMR-1B_C_DME_20230420T103323_LD_20280110T114800_
 ↪20280110T115700_TN.nc'

#dt = datetime.strptime('20230420T103323', '%Y%m%dT%H%M%S')

l1bfile = os.path.join(l1bpath, l1bfn)

pdate = datetime.strptime('20280110', '%Y%m%d')
tdate = pdate - relativedelta(years=10)
qdate = pdate + timedelta(days=1)

# Icemask data and output locations
#icemaskinputdir = 'https://thredds.met.no/thredds/dodsC/osisaf/met.no/reprocessed/
 ↪ice/conc_450a_files/{:%Y}/{:%m}'.format(tdate, tdate)
#icemaskinputfile = 'ice_conc_{}_ease2-250_cdr-v3p0_{:%Y%m%d}1200.nc'.format(hemi,␣
 ↪tdate)
#icemaskinput = cp_and_date_change_iceconc(os.path.join(icemaskinputdir,␣
 ↪icemaskinputfile), concpath, pdate)

algo_version = '0.1'

plotfigs = True
```

13.3 Step 1: Cross-correlation algorithm to find ice drift

For the Continuous Maximum Cross-Correlation algorithm, two brightness temperature gridded files, enhanced by the Laplacian algorithm and with different timestamps, are required. The algorithm matches features between these images on a fractional pixel grid.

The steps of the cross-correlation algorithm are:

1. Determination of which pixels should be included in the cross-correlation, excluding land and ocean pixels.
2. Fractional pixel cross-correlation simultaneously on the gridded swaths - forward/back scans, V and H polarisations, K and Ka channels.
3. Correction of erroneous vectors using the nearest neighbour method and creation of status flags.

Currently the C code has limited memory for channels, so the Ka-band forward and backward scans with V and H polarisations are accepted by the code. It should be possible to add K-band later.

In addition, the forward and backward scans are treated here as having the same timestamp, later it can be refined to include the 7-minute delay between these two.

```
# Copying the files with new names, to interface with the C-code

gridname = gridin.format(hemi)
chanstr = 'tb19hfw-tb19vfw-tb19hbk-tb19vbk-tb37hfw-tb37vfw-tb37hbk-tb37vbk'

dsname = os.path.join(procpath, 'bt_{}_{:%Y%m%d}.nc'.format(gridname, pdate))
dsname2 = os.path.join(procpath, 'bt_{}_{:%Y%m%d}.nc'.format(gridname, qdate))

newname1 = 'tc_wght_cimr-cimr_{}_{}_{}12.nc'.format(chanstr, gridin.format(hemi),␣
 ↪datetime.strftime(pdate, '%Y%m%d'))
newname2 = 'tc_wght_cimr-cimr_{}_{}_{}12.nc'.format(chanstr, gridin.format(hemi),␣
 ↪datetime.strftime(qdate, '%Y%m%d'))
shutil.copyfile(dsname, os.path.join(os.path.dirname(dsname), newname1))
shutil.copyfile(dsname2, os.path.join(os.path.dirname(dsname), newname2))
```

```
'/home/emilyjd/cimr-devalgo/SeaIceDrift_ATBD_v2/book/../../data/processing/tc_wght_
 ↪cimr-cimr_tb19hfw-tb19vfw-tb19hbk-tb19vbk-tb37hfw-tb37vfw-tb37hbk-tb37vbk_nh-
 ↪ease2-050_2028011112.nc'
```

```
%load_ext autoreload
%autoreload

from icedrift_wrapper import icedrift_wrapper
rad = 100.
rad_neigh = 150.
# Original rad=75. rad_neigh=125.
# Worked with rad=100. rad_neigh=150.
# rad = 25. rad_neigh=150. has lots of corrected by neighbours, but a better field
chan_list = ['tb37hfw_lap', 'tb37vfw_lap', 'tb37hbk_lap', 'tb37vbk_lap']
# Suppress the proj4 string warning on this
with warnings.catch_warnings():
    warnings.simplefilter("ignore")

    idrift = icedrift_wrapper(pdate, qdate, procpath, procpath, driftpath, os.path.
 ↪join(logpath, 'cmcc-test.log'),
```

(continues on next page)

(continued from previous page)

```
                        'cimr-cimr', gridin.format(hemi), chan_list, rad, rad_neigh,
                        area_out=gridout.format(hemi))
```

```
Daily maps found (cimr-cimr):

        Day 1 : /home/emilyjd/cimr-devalgo/SeaIceDrift_ATBD_v2/book/../../data/
↪processing/tc_wght_cimr-cimr_tb19hfw-tb19vfw-tb19hbk-tb19vbk-tb37hfw-tb37vfw-
↪tb37hbk-tb37vbk_nh-ease2-050_2028011012.nc

        Day 2 : /home/emilyjd/cimr-devalgo/SeaIceDrift_ATBD_v2/book/../../data/
↪processing/tc_wght_cimr-cimr_tb19hfw-tb19vfw-tb19hbk-tb19vbk-tb37hfw-tb37vfw-
↪tb37hbk-tb37vbk_nh-ease2-050_2028011112.nc
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[7], line 15
     12 with warnings.catch_warnings():
     13     warnings.simplefilter("ignore")
---> 15     idrift = icedrift_wrapper(pdate, qdate, procpath, procpath, driftpath,␣
↪os.path.join(logpath, 'cmcc-test.log'),
     16                               'cimr-cimr', gridin.format(hemi), chan_list, rad,
↪ rad_neigh,
     17                               area_out=gridout.format(hemi))

File ~/cimr-devalgo/SeaIceDrift_ATBD_v2/book/../../SeaIceDrift_ATBD_v2/algorithm/
↪src_sied/icedrift_wrapper.py:377, in icedrift_wrapper(start_date, end_date,␣
↪start_dir, end_dir, out_dir, c_logfile, instr, area, channels, radius, rad_neigh,
↪ area_out)
    375 lon = out_lons[i]
    376 lat = out_lats[i]
--> 377 (newrow, newcol) = img_area_def.get_xy_from_lonlat(lon, lat)
    378 try:
    379     iwcs[i] = int((newcol * img_nx) + newrow)

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/pyresample/
↪geometry.py:2340, in AreaDefinition.get_xy_from_lonlat(self, lon, lat)
   2321 """Retrieve closest x and y coordinates.
   2322
   2323 Retrieve the closest x and y coordinates (column, row indices) for the
   (...)
   2334     (x, y) : tuple of points/arrays
   2335 """
   2336 warnings.warn("'get_xy_from_lonlat' is deprecated, please use "
   2337               "'get_array_indices_from_lonlat' instead.",␣
↪DeprecationWarning,
   2338               stacklevel=2)
-> 2340 return self.get_array_indices_from_lonlat(lon, lat)

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/pyresample/
↪geometry.py:1453, in masked_ints.<locals>.wrapper(self, xm, ym)
   1449 @wraps(func)
   1450 def wrapper(self, xm, ym):
   1451     is_scalar = np.isscalar(xm) and np.isscalar(ym)
-> 1453     x__, y__ = func(self, xm, ym)
   1454     x__ = np.round(x__).astype(int)
```

(continues on next page)

(continued from previous page)

```
   1455     y__ = np.round(y__).astype(int)

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/pyresample/
↪geometry.py:2197, in AreaDefinition.get_array_indices_from_lonlat(self, lon, lat)
   2177 @masked_ints
   2178 def get_array_indices_from_lonlat(self, lon, lat):
   2179     """Find the closest integer grid cell index for a given lon/lat.
   2180
   2181     If lon,lat is a point, a ValueError is raised if it is outside the area
   (...)
   2195         ValueError: if the return point is outside the area domain
   2196     """
-> 2197     return self.get_array_coordinates_from_lonlat(lon, lat)

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/pyresample/
↪geometry.py:1489, in daskify_2in_2out.<locals>.wrapper(self, coord1, coord2)
   1486 @wraps(func)
   1487 def wrapper(self, coord1, coord2):
   1488     if da is None or not (isinstance(coord1, da.Array) or␣
↪isinstance(coord2, da.Array)):
-> 1489         return func(self, coord1, coord2)
   1490     newfunc = partial(func, self)
   1491     dims = '(' + ', '.join('i_' + str(i) for i in range(coord1.ndim)) + ')'

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/pyresample/
↪geometry.py:2139, in AreaDefinition.get_array_coordinates_from_lonlat(self, lon,␣
↪lat)
   2125 @daskify_2in_2out
   2126 def get_array_coordinates_from_lonlat(self, lon, lat):
   2127     """Retrieve the array coordinates (float) for a given lon/lat.
   2128
   2129     If lon,lat is a tuple of sequences of longitudes and latitudes, a tuple
   (...)
   2137         floats or arrays of floats: the array coordinates (cols/rows)
   2138     """
-> 2139     xm_, ym_ = self.get_projection_coordinates_from_lonlat(lon, lat)
   2140     return self.get_array_coordinates_from_projection_coordinates(xm_, ym_)

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/pyresample/
↪geometry.py:1489, in daskify_2in_2out.<locals>.wrapper(self, coord1, coord2)
   1486 @wraps(func)
   1487 def wrapper(self, coord1, coord2):
   1488     if da is None or not (isinstance(coord1, da.Array) or␣
↪isinstance(coord2, da.Array)):
-> 1489         return func(self, coord1, coord2)
   1490     newfunc = partial(func, self)
   1491     dims = '(' + ', '.join('i_' + str(i) for i in range(coord1.ndim)) + ')'

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/pyresample/
↪geometry.py:2235, in AreaDefinition.get_projection_coordinates_from_lonlat(self,␣
↪lon, lat)
   2221 @daskify_2in_2out
   2222 def get_projection_coordinates_from_lonlat(self, lon, lat):
   2223     """Get the projection coordinate from longitudes and latitudes.
   2224
   2225     If lon,lat is a tuple of sequences of longitudes and latitudes, a tuple
```

(continues on next page)

(continued from previous page)

```
   (...)
   2233         floats or arrays of floats: the projection coordinates x, y in␣
 ↪meters
   2234     """
-> 2235     p = Proj(self.crs)
   2236     return p(lon, lat)

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/pyproj/proj.
 ↪py:134, in Proj.__init__(self, projparams, preserve_units, **kwargs)
    131     projstring = self.crs.to_proj4() or self.crs.srs
    133 self.srs = re.sub(r"\s\+?type=crs", "", projstring).strip()
--> 134 super().__init__(TransformerFromPipeline(cstrencode(self.srs)))

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/pyproj/
 ↪transformer.py:326, in Transformer.__init__(self, transformer_maker)
    320     raise ProjError(
    321         "Transformer must be initialized using: "
    322         "'from_crs', 'from_pipeline', or 'from_proj'."
    323     )
    325 self._local = TransformerLocal()
--> 326 self._local.transformer = transformer_maker()
    327 self._transformer_maker = transformer_maker

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/pyproj/
 ↪transformer.py:130, in TransformerFromPipeline.__call__(self)
    124 def __call__(self) -> _Transformer:
    125     """
    126     Returns
    127     -------
    128     _Transformer
    129     """
--> 130     return _Transformer.from_pipeline(self.proj_pipeline)

File pyproj/_transformer.pyx:594, in pyproj._transformer._Transformer.from_
 ↪pipeline()

File pyproj/_transformer.pyx:328, in pyproj._transformer._Transformer._initialize_
 ↪from_projobj()

File ~/miniconda3/envs/cimr-devalgo-sid/lib/python3.9/site-packages/pyproj/
 ↪exceptions.py:19, in ProjError.clear()
     16         ProjError.clear()
     17     super().__init__(error_message)
---> 19 @staticmethod
     20 def clear() -> None:
     21     """
     22     This will clear the internal PROJ erro message.
     23     """
     24     ProjError.internal_proj_error = None

KeyboardInterrupt:
```

13.4 Step 2: Format L2 file and write to disk

The output icedrift file is processed and written, with metadata.

```
driftx = ma.asarray(idrift['drift_x'])
driftx.mask = driftx < -1e9
drifty = ma.asarray(idrift['drift_y'])
drifty.mask = drifty < -1e9
flag = ma.asarray(idrift['flag'])

dms = driftx.shape
ddx = driftx.reshape(1, dms[0], dms[1])
ddy = drifty.reshape(1, dms[0], dms[1])
stdx = ma.asarray(idrift['std_x'])
stdx.mask = stdx < -1e9
stdx = stdx.reshape(1, dms[0], dms[1])
stdy = ma.asarray(idrift['std_y'])
stdy.mask = stdy < -1e9
stdy = stdy.reshape(1, dms[0], dms[1])
dflag = ma.asarray(idrift['flag'])
dflag = flag.reshape(1, dms[0], dms[1])
```

```
reload(l2)

# Output grid
og = gridout.format(hemi)
out_area_def = parse_area_file(griddeffile, og)[0]
olons, olats = out_area_def.get_lonlats()

# Get a template L2 format (netCDF/CF) from the Tools module
ds_l2 = l2.get_CIMR_L2_template('grid', geo_def=out_area_def, add_time=[pdate.
 ↪timestamp()])

# Create a DataArray for x and y icedrift from the template
da_dx = xr.DataArray(ddx, coords=ds_l2['template'].coords, dims=ds_l2['template'].
 ↪dims,
                      attrs=ds_l2['template'].attrs, name='driftX')
da_dx.attrs['long_name'] = 'x-component of Sea Ice Drift from the CIMR ice drift␣
 ↪algorithm v{}'.format(algo_version)
da_dx.attrs['standard_name'] = 'x_component_sea_ice_drift'
da_dx.attrs['units'] = 'km'
da_dx.attrs['coverage_content_type'] = 'physicalMeasurement'
da_dx.attrs['auxiliary_variables'] = 'status_flag'
da_dy = xr.DataArray(ddy, coords=ds_l2['template'].coords, dims=ds_l2['template'].
 ↪dims,
                      attrs=ds_l2['template'].attrs, name='driftY')
da_dy.attrs['long_name'] = 'y-component of Sea Ice Drift from the CIMR ice drift␣
 ↪algorithm v{}'.format(algo_version)
da_dy.attrs['standard_name'] = 'y_component_sea_ice_drift'
da_dy.attrs['units'] = 'km'
da_dy.attrs['coverage_content_type'] = 'physicalMeasurement'
da_dy.attrs['auxiliary_variables'] = 'status_flag'

# Create a DataArray for std x and y icedrift from the template
da_stddx = xr.DataArray(stdx, coords=ds_l2['template'].coords, dims=ds_l2['template'].
 ↪dims,
```

(continues on next page)

(continued from previous page)

```
                        attrs=ds_l2['template'].attrs, name='stdX')
da_stddx.attrs['long_name'] = 'Standard deviation of x-component of Sea Ice Drift␣
 ↪from the CIMR ice drift algorithm v{}'.format(algo_version)
da_stddx.attrs['units'] = 'km'
da_stddx.attrs['coverage_content_type'] = 'auxiliaryInformation'
da_stddx.attrs['auxiliary_variables'] = 'status_flag'
da_stddy = xr.DataArray(stdy, coords=ds_l2['template'].coords, dims=ds_l2['template'].
 ↪dims,
                        attrs=ds_l2['template'].attrs, name='stdY')
da_stddy.attrs['long_name'] = 'Standard deviation of y-component of Sea Ice Drift␣
 ↪from the CIMR ice drift algorithm v{}'.format(algo_version)
da_stddy.attrs['units'] = 'km'
da_stddy.attrs['coverage_content_type'] = 'auxiliaryInformation'
da_stddy.attrs['auxiliary_variables'] = 'status_flag'

# Create a DataArray for the status flag from the template
da_flag = xr.DataArray(dflag, coords=ds_l2['template'].coords, dims=ds_l2['template'].
 ↪dims,
                        attrs=ds_l2['template'].attrs, name='status_flag')
da_flag.attrs['long_name'] = 'Status flag of Sea Ice Drift from the CIMR ice drift␣
 ↪algorithm v{}'.format(algo_version)
da_flag.attrs['units'] = 1
da_flag.attrs['coverage_content_type'] = 'auxiliaryInformation'

# Add the data arrays to the ds_l2 object
ds_l2 = ds_l2.merge(da_dx)
ds_l2 = ds_l2.merge(da_dy)
ds_l2 = ds_l2.merge(da_stddx)
ds_l2 = ds_l2.merge(da_stddy)
ds_l2 = ds_l2.merge(da_flag)

# Customize the global attributes
ds_l2.attrs['title'] = 'CIMR L2 Sea Ice Drift'
ds_l2.attrs['summary'] = 'Sea Ice Drift computed with the prototype algorithm␣
 ↪developed in the ESA CIMR DEVALGO study. The algorithm combines Ku and Ka imagery␣
 ↪channels. The product file contains the sea ice drift, its uncertainties, and␣
 ↪processing flags.'
ds_l2.attrs['l1b_file'] = os.path.basename(l1bfile)
ds_l2.attrs['algorithm_version'] = algo_version

ds_l2.attrs['creator_name'] = 'Emily Down and Thomas Lavergne'
ds_l2.attrs['creator_email'] = 'emilyjd@met.no'
ds_l2.attrs['institution'] = 'Norwegian Meteorological Institute'

# Remove the 'template' variable (we don't need it anymore)
ds_l2 = ds_l2.drop_vars('template')

# Write to file
l2_n = 'cimr_devalgo_l2_sid_{}_{}.nc'.format(og, test_card)
l2_n = os.path.join(driftpath, l2_n)
ds_l2.to_netcdf(l2_n, format='NETCDF4_CLASSIC')
print(l2_n)
```

```
/home/emilyjd/cimr-devalgo/SeaIceDrift_ATBD_v2/algorithm/../../data/icedrift/cimr_
 ↪devalgo_l2_sid_nh-ease2-250_radiometric.nc
```

13.5 Step 3: Plotting

Here an example plot of the icedrift output is made.

```
def crs_create(gridtype, hemi):

    # Define grid based on region
    if gridtype == 'polstere':
        if hemi == 'nh':
            plot_proj4_params = {'proj': 'stere',
                                 'lat_0': 90.,
                                 'lat_ts' : 70.,
                                 'lon_0': -45.0,
                                 'a': 6378273,
                                 'b': 6356889.44891}
            plot_globe = ccrs.Globe(semimajor_axis=plot_proj4_params['a'],
                                    semiminor_axis=plot_proj4_params['b'])
            plot_crs = ccrs.NorthPolarStereo(
                central_longitude=plot_proj4_params['lon_0'], globe=plot_globe)
        else:
            plot_proj4_params = {'proj': 'stere',
                                 'lat_0': -90.,
                                 'lat_ts' : -70.,
                                 'lon_0': 0.,
                                 'a': 6378273,
                                 'b': 6356889.44891}
            plot_globe = ccrs.Globe(semimajor_axis=plot_proj4_params['a'],
                                    semiminor_axis=plot_proj4_params['b'])
            plot_crs = ccrs.SouthPolarStereo(
                central_longitude=plot_proj4_params['lon_0'], globe=plot_globe)
    elif gridtype == 'ease':
        if hemi == 'nh':
            plot_crs = ccrs.LambertAzimuthalEqualArea(central_longitude=0,
                                                      central_latitude=90,
                                                      false_easting=0,
                                                      false_northing=0)
        else:
            plot_crs = ccrs.LambertAzimuthalEqualArea(central_longitude=0,
                                                      central_latitude=-90,
                                                      false_easting=0,
                                                      false_northing=0)
    else:
        raise ValueError("Unrecognised region {}".format(region))

    return(plot_crs)
```

```
def flag_arrow_col(flag, procfmt=False):

    if procfmt:
        flgfmt = 'proc'
    else:
        flgfmt = 'final'

    # Use colours from https://sashamaps.net/docs/resources/20-colors/
    fblack = '#000000'
    fmaroon = '#800000'
```

(continues on next page)

(continued from previous page)

```
    forange = '#f58231'
    fnavy = '#000075'
    fblue = '#4363d8'
    flavender = '#dcbeff'
    fgrey = '#a9a9a9'

    fbrown = '#9A6324'
    fteal = '#469990'
    fgreen = '#3cb44b'
    fcyan = '#42d4f4'
    fmagenta = '#f032e6'

    fred = '#e6194B'
    fpurple = '#911eb4'

    flag_cols = {}
    flag_cols['final'] = {30: fblack,    # Nominal quality
                          20: fbrown,    # Single-sensor, with smaller pattern
                                         # block
                          21: forange,   # Single-sensor, with neighbours as
                                         # constraint
                          22: fmaroon,   # Interpolated
                          23: fcyan,     # Gap filling in wind drift
                          24: fteal,     # Vector replaced by wind drift
                          25: fnavy,     # Blended satellite and wind drift
}
    flag_cols['proc'] = {0: fblack,      # Nominal quality
                         16: fpurple,    # Interpolated
                         13: fred,       # Single-sensor, with neighbours as
                                         # constraint
                         17: fgreen,     # Single-sensor, with smaller
                                         # pattern block
}

    default = 'black'
    if flag in flag_cols[flgfmt].keys():
        return flag_cols[flgfmt][flag]
    else:
        return default
```

```
scale = 1000.
pc = ccrs.PlateCarree()

def plotdriftarr(ax, plot_crs, data_crs, lon, lat, dx, dy, sflag, driftflags=True):

    for i in range(dx.size):
        try:
            x0, y0 = plot_crs.transform_point(lon[~dx.mask][i], lat[~dx.mask][i], src_
 ↪crs=pc)
            adx = dx[~dx.mask][i]
            ady = dy[~dy.mask][i]
            len_arrow = math.sqrt(adx**2 + ady**2)

            # Calculate the endpoints and therefore dx, dy components of the drift␣
 ↪arrows in the plot
            # coordinate system
```

(continues on next page)

(continued from previous page)

```
            xorig, yorig = data_crs.transform_point(lon[~dx.mask][i], lat[~dx.
↪mask][i], src_crs=pc)
            xarr = xorig + adx
            yarr = yorig + ady
            x1, y1 = plot_crs.transform_point(xarr, yarr, src_crs=data_crs)
            pdx = (x1 - x0) * scale
            pdy = (y1 - y0) * scale

            # Set the colour of the drift arrows if this should be
            # done with status flags
            if driftflags:
                myflag = sflag[~dx.mask][i]
                ar_col = flag_arrow_col(myflag, procfmt=True)
            else:
                ar_col = 'black'

            # If the arrow is too small, mark a symbol instead
            if len_arrow * scale < 2000:
                plt.plot(x0, y0, 's', color=ar_col, markersize=1)
            else:
                head_length = 0.3 * len_arrow * scale
                plt.arrow(x0, y0, pdx, pdy, color=ar_col,
                    shape='full', head_length=head_length,
                    head_width=15000,
                    fill=True, length_includes_head=True,
                    width=4000)

        except:
            pass # Outside the range of points
```

```
# Plotting the ice drift with arrows
# Modified from the SeaSurfaceTemperature_ATBD_v2, by Emy Alerskans

# Drift data
xydata = {'dx': driftx, 'dy': drifty}
limminxy = np.nanmin([np.nanmin(a) for a in xydata.values()])
limminxy = limminxy - 0.1 * abs(limminxy)
limmaxxy = np.nanmax([np.nanmax(a) for a in xydata.values()])
limmaxxy = limmaxxy + 0.1 * abs(limmaxxy)
limminflag = np.nanmin(flag)
limmaxflag = np.nanmax(flag)

# Output lat/lons
og = gridout.format(hemi)
out_area_def = parse_area_file(griddeffile, og)[0]
olons, olats = out_area_def.get_lonlats()

# Coordinate reference systems
plot_crs = crs_create(gridtype, hemi)
pc = ccrs.PlateCarree()
if 'ease' in gridout:
    data_crs = crs_create('ease', hemi)
else:
    data_crs = crs_create('polstere', hemi)

# Plotting drift
```

(continues on next page)

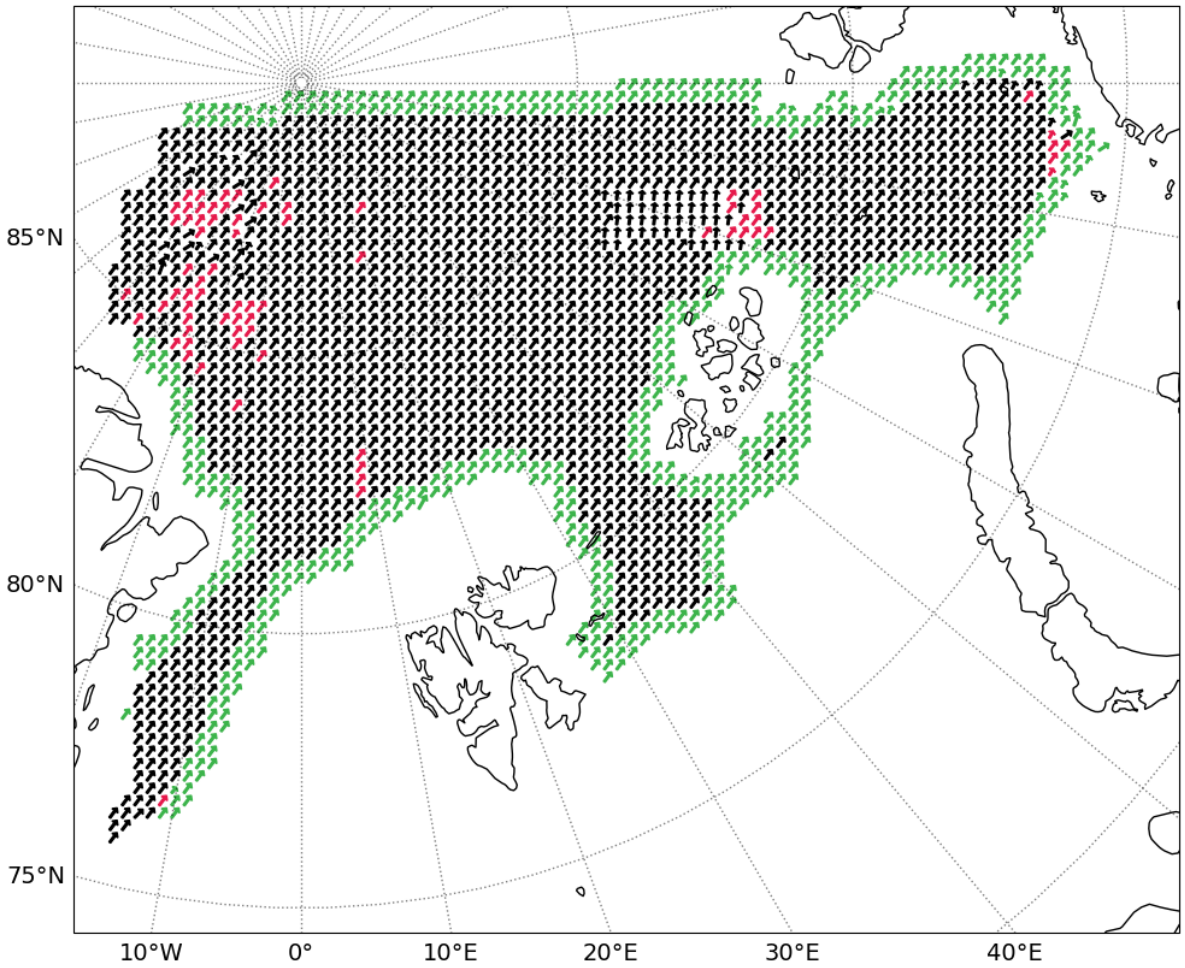
(continued from previous page)

```
fig = plt.figure(figsize=[12, 10])
ax = fig.add_subplot(1, 1, 1, projection=plot_crs)

plotdriftarr(ax, plot_crs, data_crs, olons[slo[0]:slo[1], slo[2]:slo[3]],␣
↪olats[slo[0]:slo[1], slo[2]:slo[3]],
             driftx[slo[0]:slo[1], slo[2]:slo[3]], drifty[slo[0]:slo[1],␣
↪slo[2]:slo[3]],
             flag[slo[0]:slo[1], slo[2]:slo[3]], driftflags=True)

ax.coastlines()
ax.set_extent([lon_min, lon_max, lat_min, lat_max])
gl = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True, linewidth=1, color='k',␣
↪alpha=0.5, linestyle=':')
gl.top_labels = False
gl.right_labels = False
gl.xlocator = mticker.FixedLocator(np.arange(-180, 180, lon_step))
gl.ylocator = mticker.FixedLocator(np.arange(-90, 90, lat_step))
gl.xlabel_style = {'size': 14}
gl.ylabel_style = {'size': 14}

if plotfigs:
    plt.savefig(os.path.join(figpath, 'drift_rad{}.png'.format(int(rad))))
plt.show()

# Status flags
fig = plt.figure(figsize=[12, 10])
ax = fig.add_subplot(1, 1, 1, projection=plot_crs)
im = ax.pcolormesh(olons[slo[0]:slo[1], slo[2]:slo[3]], olats[slo[0]:slo[1],␣
↪slo[2]:slo[3]],
                   flag[:][slo[0]:slo[1], slo[2]:slo[3]], transform=pc,
                   cmap=cmap, vmin=limminflag, vmax=limmaxflag)
ax.coastlines()
ax.set_extent([lon_min, lon_max, lat_min, lat_max])
gl = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True, linewidth=1, color='k',␣
↪alpha=0.5, linestyle=':')
gl.top_labels = False
gl.right_labels = False
gl.xlocator = mticker.FixedLocator(np.arange(-180, 180, lon_step))
gl.ylocator = mticker.FixedLocator(np.arange(-90, 90, lat_step))
gl.xlabel_style = {'size': 14}
gl.ylabel_style = {'size': 14}

# Colourbar
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="3%", pad="2%", axes_class=plt.Axes)
cb = fig.colorbar(im, cax=cax)
cb.set_label(label="Flag", fontsize=16, labelpad=20.0)
cb.ax.set_ylim(limminflag, limmaxflag)

print("\
FLAG VALUES \n\
Unprocessed pixel                              -1\n\
Nominal                                        0\n\
Outside image border                           1\n\
Close to image border                          2\n\
Pixel center over land                         3\n\
```

(continues on next page)

(continued from previous page)

```
No ice                                          4\n\
Close to coast or edge                          5\n\
Close to missing pixel                          6\n\
Close to unprocessed pixel                      7\n\
Icedrift optimisation failed                    8\n\
Icedrift failed                                 9\n\
Icedrift with low correlation                   10\n\
Icedrift calculation took too long              11\n\
Icedrift calculation refused by neighbours      12\n\
Icedrift calculation corrected by neighbours    13\n\
Icedrift no average                             14\n\
Icedrift masked due to summer season            15\n\
Icedrift multi-oi interpolation                 16\n\
Icedrift calcuated with smaller pattern         17\n\
Icedrift masked due to NWP                      18\n\
")
```

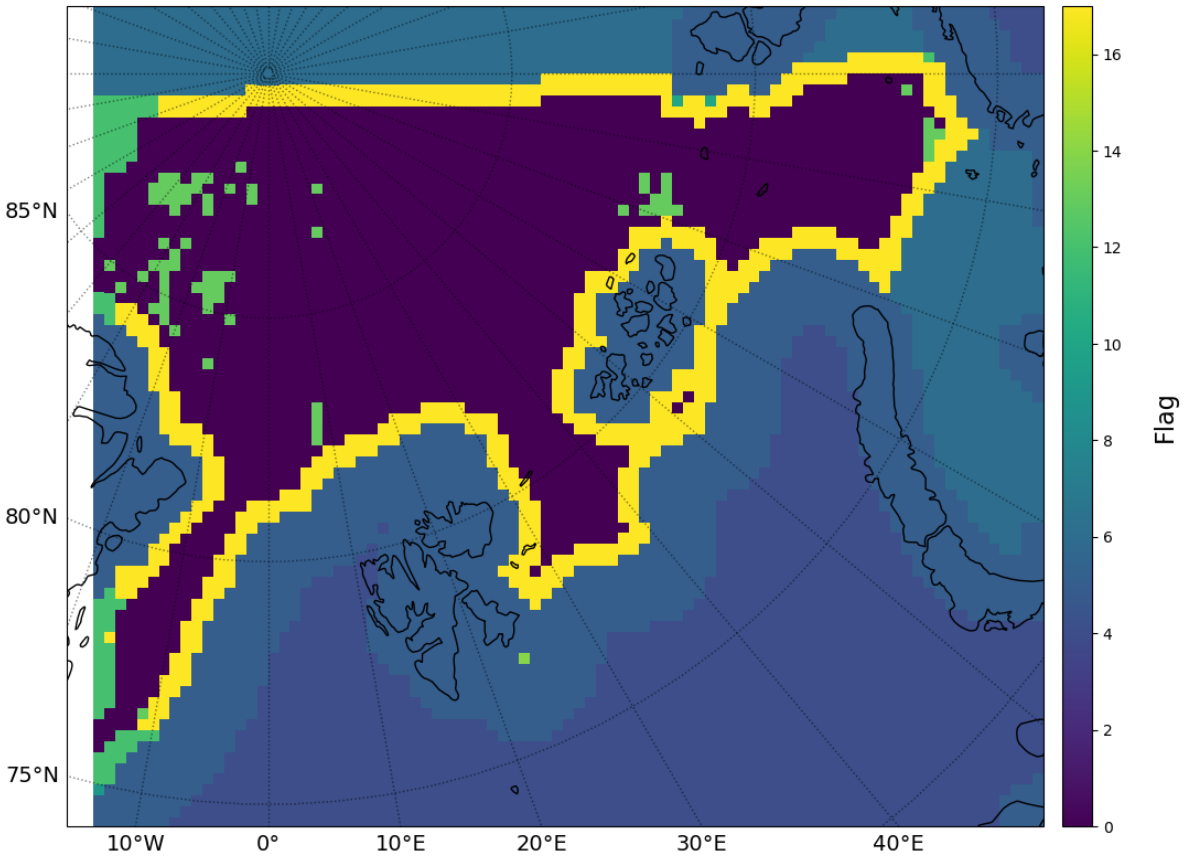



```
FLAG VALUES
Unprocessed pixel                               -1
Nominal                                         0
Outside image border                            1
```

(continues on next page)

(continued from previous page)

```
Close to image border                          2
Pixel center over land                         3
No ice                                         4
Close to coast or edge                         5
Close to missing pixel                         6
Close to unprocessed pixel                     7
Icedrift optimisation failed                   8
Icedrift failed                                9
Icedrift with low correlation                  10
Icedrift calculation took too long             11
Icedrift calculation refused by neighbours     12
Icedrift calculation corrected by neighbours   13
Icedrift no average                            14
Icedrift masked due to summer season           15
Icedrift multi-oi interpolation                16
Icedrift calcuated with smaller pattern        17
Icedrift masked due to NWP                     18
```



CHAPTER

FOURTEEN

PERFORMANCE ASSESSMENT

CHAPTER

FIFTEEN

LEVEL-2 SEA ICE DRIFT (SID) ALGORITHM FOR CIMR

This notebook implements a prototype for a Level-2 SIED algorithm for the CIMR mission.

We refer to the corresponding ATBD and especially the Baseline Algorithm Definition.

In particular, the figure below illustrates the overall concept of the processing:

15.1 Settings

Imports and general settings

```
# Paths

# Getting the path of the notebook (NOTE: not totally safe)
# The paths assume that there is an umbrella CIMR directory (of any name) containing␣
↪SeaIceDrift_ATBD_v2/ ,
# the CIMR Tools/ directory, and a directory data/L1B/ containing the L1B data, and␣
↪data/conc/ containing
# a concentration file.
# Additionally, this code expects that the CIMR L2 Sea Ice Drift algorithm notebook␣
↪has been run, to create
# the drift file
import os
cpath = os.path.join(os.getcwd(), '../..')
algpath = os.path.join(cpath, 'SeaIceDrift_ATBD_v2/algorithm/src_sied')
driftpath = os.path.join(cpath, 'data/icedrift')
figpath = os.path.join(cpath, 'data/figs')
```

```
# Imports

import sys
import math
import numpy as np
import numpy.ma as ma
from netCDF4 import Dataset
from matplotlib import pylab as plt
import matplotlib.ticker as mticker
from mpl_toolkits.axes_grid1.axes_divider import make_axes_locatable
import matplotlib.cm as cm
import cmocean
import cartopy
import cartopy.crs as ccrs
```

(continues on next page)

(continued from previous page)

```
from pyresample import parse_area_file

# Local modules contain software code that implement the SIED algorithm
if algpath not in sys.path:
    sys.path.insert(0, algpath)
```

```
# Plot settings

import matplotlib
matplotlib.rc('xtick', labelsize=10)
matplotlib.rc('ytick', labelsize=10)
matplotlib.rcParams.update({'font.size': 12})
matplotlib.rcParams.update({'axes.labelsize': 12})

font = {'family' : 'sans',
        'weight' : 'normal',
        'size'   : 12}

matplotlib.rc('font', **font)

cmap = cm.viridis

gridtype = 'ease'
gridout = '{}-ease2-250'
# Some region parameters hard-coded to show only the relevant region
# Overall shape of input grid (4320, 4320)
slo = (200, 290, 200, 290)

# EASE plotting region
lon_min = -15
lon_max = 95
lat_min = 74
lat_max = 90

# Settings for gridlines
lon_step = 10
lat_step = 5

plotfigs=True
```

15.2 Parametrize the run

User-set parameters for the running of the whole notebook

```
hemi = 'nh'

test_card = "radiometric"

griddeffile = os.path.join(algpath, 'grids_py.def')

ncfile = os.path.join(driftpath, 'cimr_devalgo_l2_sid_{}_{}.nc'.format(gridout.
 ↪format(hemi), test_card))
```

(continues on next page)

(continued from previous page)

```
pixshx = 3
pixshy = 4
```

15.3 Simple Performance Assessment

A simple performance assessment is conducted here for the prototype SID algorithm (see notebooks for pre-processing and algorithm) run on the Radiometric scene. The SID algorithm requires two scenes, with a displacement between them. We thus created a second gridded file by applying a 24-hour time displacement in the metadata, and a 3-pixel x-displacement and a 4-pixel y-displacement to all pixels. The SID was then calculated using the Continuous Maximum Cross-Correlation method between these two gridded files and is here compared to the shift values applied to create the second gridded file. This validation therefore excludes inaccuracies that may arise from the gridding phase, as well as such technical uncertainties such as geolocation.

The chosen 3-pixel displacement in x over a 24-hour period is equivalent in the 5km input grid to 15km/day ice drift, and 4-pixel displacement in y is equivalent to a 20 km/day ice drift. These are reasonably realistic SID values.

```
def crs_create(gridtype, hemi):

    # Define grid based on region
    if gridtype == 'polstere':
        if hemi == 'nh':
            plot_proj4_params = {'proj': 'stere',
                                 'lat_0': 90.,
                                 'lat_ts' : 70.,
                                 'lon_0': -45.0,
                                 'a': 6378273,
                                 'b': 6356889.44891}
            plot_globe = ccrs.Globe(semimajor_axis=plot_proj4_params['a'],
                                    semiminor_axis=plot_proj4_params['b'])
            plot_crs = ccrs.NorthPolarStereo(
                central_longitude=plot_proj4_params['lon_0'], globe=plot_globe)
        else:
            plot_proj4_params = {'proj': 'stere',
                                 'lat_0': -90.,
                                 'lat_ts' : -70.,
                                 'lon_0': 0.,
                                 'a': 6378273,
                                 'b': 6356889.44891}
            plot_globe = ccrs.Globe(semimajor_axis=plot_proj4_params['a'],
                                    semiminor_axis=plot_proj4_params['b'])
            plot_crs = ccrs.SouthPolarStereo(
                central_longitude=plot_proj4_params['lon_0'], globe=plot_globe)
    elif gridtype == 'ease':
        if hemi == 'nh':
            plot_crs = ccrs.LambertAzimuthalEqualArea(central_longitude=0,
                                                      central_latitude=90,
                                                      false_easting=0,
                                                      false_northing=0)
        else:
            plot_crs = ccrs.LambertAzimuthalEqualArea(central_longitude=0,
                                                      central_latitude=-90,
                                                      false_easting=0,
                                                      false_northing=0)
```

(continues on next page)

(continued from previous page)

```
    else:
        raise ValueError("Unrecognised region {}".format(region))

    return(plot_crs)
```

```
# Reading NetCDF file

def nc_read(ncfile, var, skip=None, procfmt=False):

    ncdata = {}

    if isinstance(var, list):
        v = var[0]
    else:
        v = var

    # Reading from the NetCDF file
    with Dataset(ncfile, 'r') as dataset:

        try:
            ncdata['fv'] = dataset.variables[v].__dict__['_FillValue']
        except:
            pass

        if isinstance(var, list):
            varlist = var
        else:
            varlist = [var]
        for item in varlist:
            vardata = dataset[item][:]
            if len(vardata.shape) == 3:
                if skip:
                    vardata = vardata[0, ::skip, ::skip]
                else:
                    vardata = vardata[0, :, :]
            else:
                if skip:
                    vardata = vardata[::skip, ::skip]
                else:
                    vardata = vardata[:, :]

            # This is for reducing flags to the lowest integers
            if var in ['statusflag', 'status_flag', 'flag']:
                vardata = np.asarray(vardata, float) + 100
                uniques = np.unique(vardata)
                uniques = uniques[np.logical_not(np.isnan(uniques))]
                for newval, origval in enumerate(uniques):
                    vardata[vardata == origval] = newval
                ncdata['sf_labs'] = [str(int(u - 100)) for u in uniques]

            # NOTE: Be very careful with the fill value here. Trying to
            # use ncdata['fv'] as the fill_value for a status array such
            # as 'flag' means that flag values of 0 (i.e. nominal) are
            # masked out
            if 'fv' in ncdata.keys() and item not in ['statusflag',
                                                      'status_flag', 'flag']:
```

(continues on next page)

(continued from previous page)

```
                ncdata[item] = ma.array(vardata, fill_value=ncdata['fv'])
            else:
                ncdata[item] = ma.array(vardata)

            if var in ['statusflag', 'status_flag', 'flag']:
                ncdata[item].mask = None
            else:
                ncdata[item].mask = ncdata[item].data == ncdata[item].fill_value

        if skip:
            ncdata['lon'] = dataset.variables['lon'][::skip, ::skip]
            ncdata['lat'] = dataset.variables['lat'][::skip, ::skip]
        else:
            ncdata['lon'] = dataset.variables['lon'][:]
            ncdata['lat'] = dataset.variables['lat'][:]

        # Try fetching time info
        try:
            try:
                d0 = datetime.strptime(dataset.start_date,'%Y-%m-%d %H:%M:%S')
                d1 = datetime.strptime(dataset.stop_date,'%Y-%m-%d %H:%M:%S')
            except:
                try:
                    d0 = datetime.strptime(dataset.start_date_and_time,
                                           '%Y-%m-%dT%H:%M:%SZ')
                    d1 = datetime.strptime(dataset.end_date_and_time,
                                           '%Y-%m-%dT%H:%M:%SZ')
                except:
                    d0 = datetime.strptime(dataset.time_coverage_start,
                                           '%Y-%m-%dT%H:%M:%SZ')
                    d1 = datetime.strptime(dataset.time_coverage_end,
                                           '%Y-%m-%dT%H:%M:%SZ')
            d0_00 = datetime.combine(d0.date(),time(0))
            d1_00 = datetime.combine(d1.date(),time(0))
            ncdata['sdate'] = d0_00
            ncdata['edate'] = d1_00
            ncdata['tspan_hours'] = (d1_00 - d0_00).total_seconds() / (60.*60.)
        except:
            pass

        ncdata['time'] = dataset.variables['time'][0]

    return ncdata
```

```
# Reading in the data

varlist = ['driftX', 'driftY', 'status_flag']
iddata = nc_read(ncfile, varlist, skip=None, procfmt=True)
```

```
# Plotting the ice drift
# Modified from the SeaSurfaceTemperature_ATBD_v2, by Emy Alerskans

# Drift data
xydata = {'dx': iddata['driftX'], 'dy': iddata['driftY']}
flag = iddata['status_flag']
```

(continues on next page)

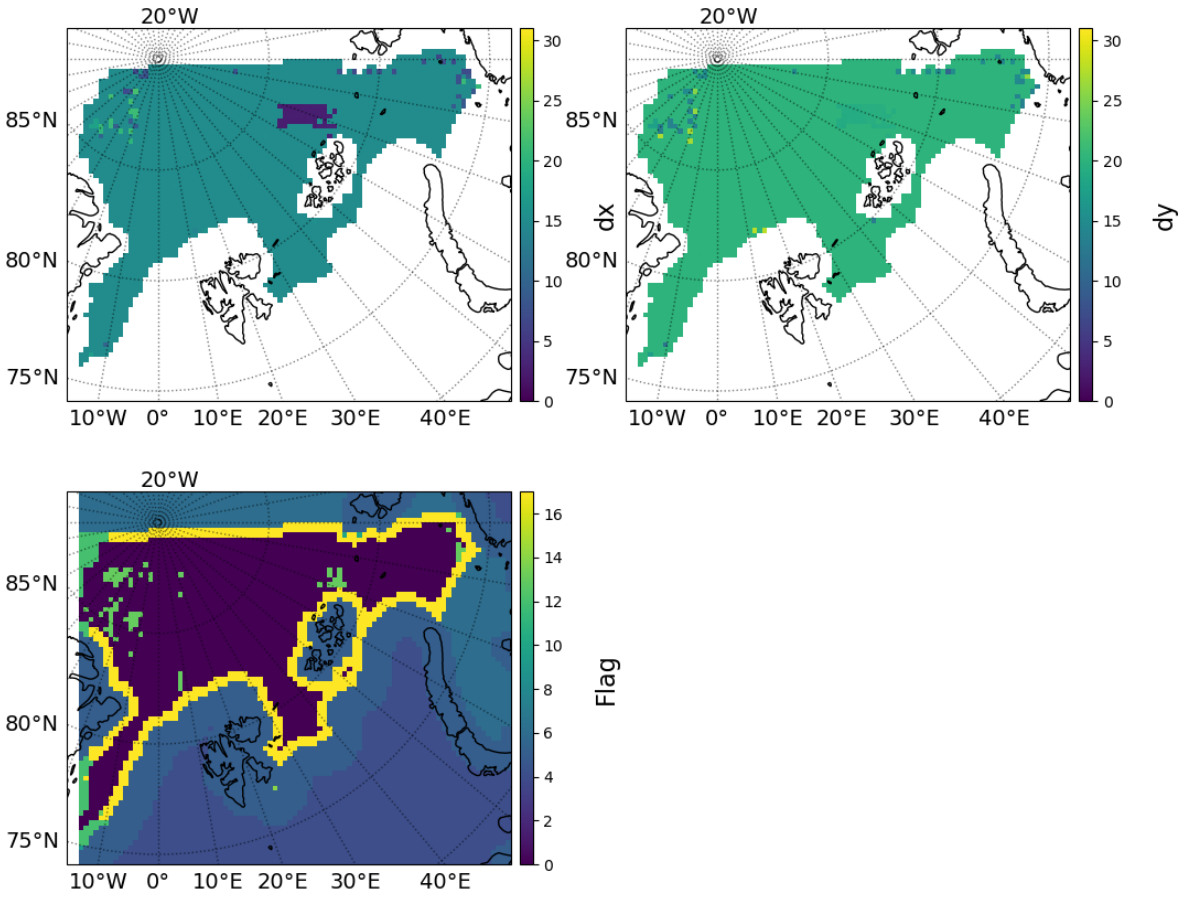
(continued from previous page)

```
limminxy = np.nanmin([np.nanmin(a) for a in xydata.values()])
limminxy = limminxy - 0.1 * abs(limminxy)
limmaxxy = np.nanmax([np.nanmax(a) for a in xydata.values()])
limmaxxy = limmaxxy + 0.1 * abs(limmaxxy)
limminflag = np.nanmin(flag)
limmaxflag = np.nanmax(flag)

# Output lat/lons
og = gridout.format(hemi)
out_area_def = parse_area_file(griddeffile, og)[0]
olons, olats = out_area_def.get_lonlats()

# Coordinate reference systems
plot_crs = crs_create(gridtype, hemi)
pc = ccrs.PlateCarree()

# Plotting drift
fig = plt.figure(figsize=[12, 10])
for i, var in enumerate(list(xydata.keys())):
    ax = fig.add_subplot(2, 2, i+1, projection=plot_crs)
    im = ax.pcolormesh(olons[slo[0]:slo[1], slo[2]:slo[3]], olats[slo[0]:slo[1],␣
 ↪slo[2]:slo[3]],
                       xydata[var][:][slo[0]:slo[1], slo[2]:slo[3]], transform=pc,
                       cmap=cmap, vmin=0, vmax=math.ceil(limmaxxy))
    ax.coastlines()
    ax.set_extent([lon_min, lon_max, lat_min, lat_max])
    gl = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True, linewidth=1, color='k
 ↪', alpha=0.5, linestyle=':')
    gl.top_labels = False
    gl.right_labels = False
    gl.xlocator = mticker.FixedLocator(np.arange(-180, 180, lon_step))
    gl.ylocator = mticker.FixedLocator(np.arange(-90, 90, lat_step))
    gl.xlabel_style = {'size': 14}
    gl.ylabel_style = {'size': 14}

    # Colourbar
    divider = make_axes_locatable(ax)
    cax = divider.append_axes("right", size="3%", pad="2%", axes_class=plt.Axes)
    cb = fig.colorbar(im, cax=cax)
    cb.set_label(label=var, fontsize=16, labelpad=20.0)
    cb.ax.set_ylim(0, math.ceil(limmaxxy))


# Status flags
ax = fig.add_subplot(2, 2, 3, projection=plot_crs)
im = ax.pcolormesh(olons[slo[0]:slo[1], slo[2]:slo[3]], olats[slo[0]:slo[1],␣
 ↪slo[2]:slo[3]],
                   flag[:][slo[0]:slo[1], slo[2]:slo[3]], transform=pc,
                   cmap=cmap, vmin=limminflag, vmax=limmaxflag)
ax.coastlines()
ax.set_extent([lon_min, lon_max, lat_min, lat_max])
gl = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True, linewidth=1, color='k',␣
 ↪alpha=0.5, linestyle=':')
gl.top_labels = False
gl.right_labels = False
gl.xlocator = mticker.FixedLocator(np.arange(-180, 180, lon_step))
```

(continues on next page)

(continued from previous page)

```
gl.ylocator = mticker.FixedLocator(np.arange(-90, 90, lat_step))
gl.xlabel_style = {'size': 14}
gl.ylabel_style = {'size': 14}

# Colourbar
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="3%", pad="2%", axes_class=plt.Axes)
cb = fig.colorbar(im, cax=cax)
cb.set_label(label="Flag", fontsize=16, labelpad=20.0)
cb.ax.set_ylim(limminflag, limmaxflag)

if plotfigs:
    plt.savefig(os.path.join(figpath, 'dxdy.png'))
plt.show()

print("\
FLAG VALUES \n\
Unprocessed pixel                               -1\n\
Nominal                                         0\n\
Outside image border                            1\n\
Close to image border                           2\n\
Pixel center over land                          3\n\
No ice                                          4\n\
Close to coast or edge                          5\n\
Close to missing pixel                          6\n\
Close to unprocessed pixel                      7\n\
Icedrift optimisation failed                    8\n\
Icedrift failed                                 9\n\
Icedrift with low correlation                   10\n\
Icedrift calculation took too long              11\n\
Icedrift calculation refused by neighbours      12\n\
Icedrift calculation corrected by neighbours    13\n\
Icedrift no average                             14\n\
Icedrift masked due to summer season            15\n\
Icedrift multi-oi interpolation                 16\n\
Icedrift calcuated with smaller pattern         17\n\
Icedrift masked due to NWP                      18\n\
")
```



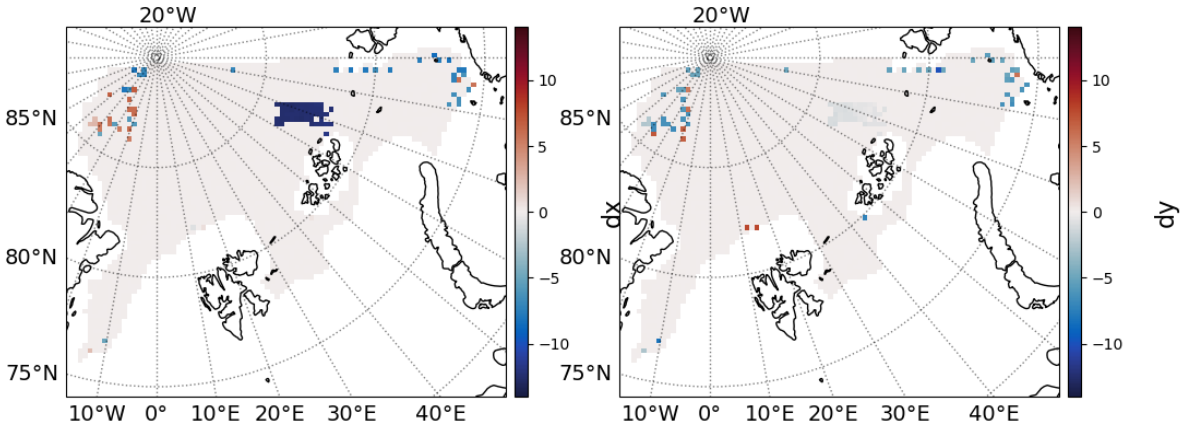
FLAG VALUES	
Unprocessed pixel	-1
Nominal	0
Outside image border	1
Close to image border	2
Pixel center over land	3
No ice	4
Close to coast or edge	5
Close to missing pixel	6
Close to unprocessed pixel	7
Icedrift optimisation failed	8
Icedrift failed	9
Icedrift with low correlation	10
Icedrift calculation took too long	11
Icedrift calculation refused by neighbours	12
Icedrift calculation corrected by neighbours	13
Icedrift no average	14
Icedrift masked due to summer season	15
Icedrift multi-oi interpolation	16
Icedrift calcuated with smaller pattern	17
Icedrift masked due to NWP	18

Here the output x- and y-components of drift are plotted. They are close to the 15 km/day in x and 20 km/day in y expected from the input data, except for one patch around 70E, 83N, which the ice drift algorithm has not calculated correctly for the x-direction. This is suspected to be due to the limitations of the Radiometric test card, which has somewhat “blocky” features, and for the next round of algorithm improvements emphasis will be on an improved set of test scenes. These

are planned to be created from model data, transformed onto a set of swaths separated in time, and then run through the CIMR data simulator. Therefore these scenes will have the texture expected of satellite observations of sea ice at this resolution, and the algorithm will have more pattern to match.

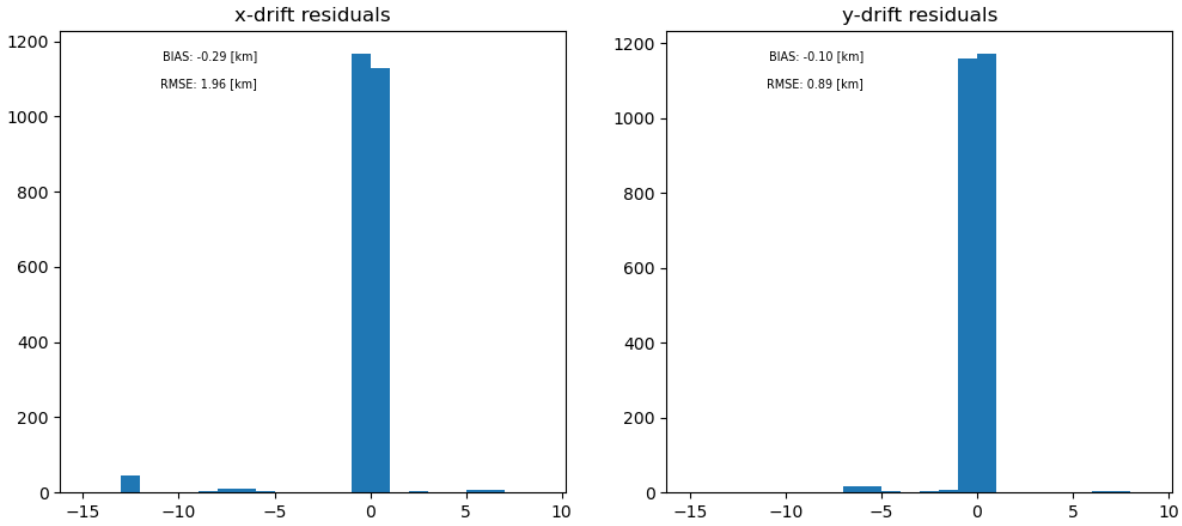
There is also a region around 20W,86N where the vectors were interpolated and are thus less accurate.

```
# Plotting the residuals
# Modified from the SeaSurfaceTemperature_ATBD_v2, by Emy Alerskans

pixsize = 5.0
expectedx = pixshx * pixsize
expectedy = pixshy * pixsize
testx = np.full(iddata['driftX'].shape, expectedx)
testy = np.full(iddata['driftX'].shape, expectedy)

testdata = {'dx': testx, 'dy': testy}
resx = iddata['driftX'] - testx
resy = iddata['driftY'] - testy
resdata = {'dx': resx, 'dy': resy}

limminres = np.nanmin([np.nanmin(a) for a in resdata.values()])
limminres = limminres - 0.1 * abs(limminres)
limmaxres = np.nanmax([np.nanmax(a) for a in resdata.values()])
limmaxres = limmaxres + 0.1 * abs(limmaxres)
lim = max([abs(limminres), limmaxres])

fig = plt.figure(figsize=[12, 5])
for i, var in enumerate(list(resdata.keys())):
    ax = fig.add_subplot(1, 2, i+1, projection=plot_crs)
    im = ax.pcolormesh(olons[slo[0]:slo[1], slo[2]:slo[3]], olats[slo[0]:slo[1],␣
 ↪slo[2]:slo[3]],
                       resdata[var][:][slo[0]:slo[1], slo[2]:slo[3]], transform=pc,
                       cmap=cmocean.cm.balance, vmin=-lim, vmax=lim)
    ax.coastlines()
    ax.set_extent([lon_min, lon_max, lat_min, lat_max])
    gl = ax.gridlines(crs=ccrs.PlateCarree(), draw_labels=True, linewidth=1, color='k
 ↪', alpha=0.5, linestyle=':')
    gl.top_labels = False
    gl.right_labels = False
    gl.xlocator = mticker.FixedLocator(np.arange(-180, 180, lon_step))
    gl.ylocator = mticker.FixedLocator(np.arange(-90, 90, lat_step))
    gl.xlabel_style = {'size': 14}
    gl.ylabel_style = {'size': 14}

    # Colourbar
    divider = make_axes_locatable(ax)
    cax = divider.append_axes("right", size="3%", pad="2%", axes_class=plt.Axes)
    cb = fig.colorbar(im, cax=cax)
    cb.set_label(label=var, fontsize=16, labelpad=20.0)
    cb.ax.set_ylim(-lim, lim)

if plotfigs:
    plt.savefig(os.path.join(figpath, 'resdxdy.png'))
plt.show()
```



Here the residuals are plotted, and again, the areas in which the algorithm did not fully succeed are highlighted, while most of the field has a well-calculated ice drift.

```
# Plotting a histogram

minbin = math.floor(limminres)
maxbin = math.ceil(limmaxres) + 1
hbins = list(range(minbin, maxbin, 1))
#step = 0.5
#hbins = [(step * x) + minbin for x in range((int(abs(maxbin - minbin) / step)))]

import matplotlib.transforms as transforms

fig = plt.figure(figsize=(12,5))
ax = {}
c = {}
for i, var in enumerate(list(resdata.keys())):
    ax[i] = fig.add_subplot(1,len(resdata),i+1)
    ax[i].hist(resdata[var][:].flatten(), bins=hbins)
    ax[i].set_title('{}-drift residuals'.format(var[1:2]), fontsize=12)

    ax[i].text(0.39,0.96, 'BIAS: {:.2f} [km]'.format(np.nanmean(resdata[var])),
        transform=ax[i].transAxes, ha='right', va='top', fontsize='xx-small')
    ax[i].text(0.39,0.90, 'RMSE: {:.2f} [km]'.format(np.nanstd(resdata[var])),
        transform=ax[i].transAxes, ha='right', va='top', fontsize='xx-small')

if plotfigs:
    plt.savefig(os.path.join(figpath, 'hist_res.png'))
plt.show()
```



The histograms of residuals are shown here. The RMSE for this artifical data is below the target value of 2.6 km/day from the MRD, ~2 km (lower for y-direction). The bias is also reasonably low, -0.3 km (lower for y-direction). The mismatch between x and y here is due to the fact that the bulk of the error comes from the one single patch that the algorithm did not calculate well in the x-direction. It is predicted that this will not be an issue with more realistic test scenes.

In addition, the residuals here are not expected to be accurate, due to the lack of natural features in the test card, and the fact that the second gridded image was exactly the first gridded image but with a shift. We will be able to characterise likely residuals better with the more realistic test cards we are planning to use in L2PAD.

CHAPTER

SIXTEEN

ROADMAP FOR FUTURE ATBD DEVELOPMENT

The SID ATBD was prepared by gathering text and equations from a number of existing documentations, for example from the EUMETSAT OSI SAF or CIMR MRC. This was greatly facilitated by the similarities between the LaTeX and Markdown syntax for mathematical expressions.

In terms of algorithm, we continue the approach introduced in CIMR MRC to do Level-2 "swath-to-swath" retrievals of sea-ice motion, instead of the classical approach to prepare sea-ice motion products at Level-3 from daily aggregated maps of brightness temperatures. Several characteristics of the L2 SID product will put specific requirements on the future CIMR ground segment: it is one of the only products that will start with a remapping of the L1B TB onto an EASE2 grid as sea-ice drift cannot be processed in swath projection. In addition, sea-ice drift cannot be computed from a single incoming L1B swath but requires two swaths: the incoming swath, and a previous swath that is accessible to the processing and stored from a previous run. If multiple processors or computing nodes are available for the SID processor, one can prepare several sea-ice drift products for each incoming swath, to fully exploit the frequent revisit time of the CIMR mission.

Since the core SID algorithm has been long used in other projects, and has already been investigated for "swath-to-swath use", the future development work is therefore largely focused on the niche requirements for CIMR test cards needed to thoroughly validate this algorithm. There are two technical points in the algorithm itself which will be investigated during the L2PAD project. Firstly, the core cross-correlation algorithm used for the calculation of SID optimizes multiple channels at once. This is currently able to optimize up to 6 channels simultaneously, however, since we require 8 channels (Ku-band and Ka-band, V and H-polarizations, and forward and back scans in all permutations), this code needs expansion to ensure that it can run without encountering memory issues. The current demonstration uses only Ka-band. Secondly, there is a time delay of around ~5 minutes between the forward and backward scans of CIMR (at the center of the swaths). This is currently ignored as being significantly less than the 24-hour period used for the manufactured test data, however, when we investigate shorter time delays between swaths, this short time delay will become more significant, so we will investigate if this should be corrected to improve accuracy of the calculated drift.

A significant part of the work for the next phase will be to investigate possible separation times between the incoming swaths, and which swaths should be paired to give the most accurate ice drift with the shortest waiting time. The longer the period between two swaths, the easier it is to extract accurate sea-ice motion as the ice features have travelled a longer distance. However, since the trajectories of the sea ice are not measured, but instead the displacement vector between the two timepoints, the shorter the delay between each measurement, the better picture overall can be built up of the motion, given many successive measurements. In addition, we wish to provide users with as recent information as possible, approaching a near-real time option. It is not realistic to pair each and every swath with every other, given the large number per 24-hour period, so we need to determine the optimal swath pairings.

The biggest limitation with the current performance assessment is the test data available. Since we must use two swaths separated in time to calculate the ice drift, we require this kind of test data. The algorithm cross-correlates sea-ice features between two swaths, and so we also require reasonably physical sea-ice texture in the test scenes. We saw in the initial performance assessment that the algorithm gave "blocky" results due to the blockiness of the Radiometric test card. The plan for L2PAD is to obtain model data of brightness temperature and ice drift for a set of timestamps, for example from neXtSIM, to grid this model data onto simulated CIMR swaths according to the timestamps, and then to use these data as input to create simulated CIMR swath test scenes. This will allow us to make realistic validations, ideally on a range of different time delays between swath pairs.

CHAPTER

SEVENTEEN

REFERENCES

BIBLIOGRAPHY

[1] T. A. Agnew, Hao Le, and T. Hirose. Estimation of large scale sea ice motion from SSMI 85.5 GHz imagery. *Annals of Glaciology*, 24:305–311, 1997.

[2] M. J. Brodzik, B. Billingsley, T. Haran, B. Raup, and M. H. Savoie. EASE-Grid 2.0: incremental but significant improvements for earth-gridded data sets. *ISPRS International Journal of Geo-Information*, 1(1):32 – 45, 2012. doi:10.3390/ijgi1010032.

[3] W. J. Emery, C. W. Fowler, J. Hawkins, and R. H. Preller. Fram Strait satellite image-derived ice motions. *Journal of Geophysical Research*, 96(C3):4751–4768, March 1991.

[4] W. J. Emery, A. C. Thomas, M. J. Collins, W. R. Crawford, and D. L. Mackas. An objective method for computing advective surface velocities from sequential infrared satellite images. *Journal of Geophysical Research*, 91:12865–12878, 1986.

[5] Robert Ezraty, Fanny Girard-Ardhuin, and D. Croizé-Fillon. Sea ice drift in the central Arctic estimated using the 89 GHz brightness temperatures of the Advanced Microwave Scanning Radiometer – User's manual. v2.0, CERSAT, IFREMER, France, February 2007.

[6] Fanny Girard-Ardhuin and Robert Ezraty. Enhanced arctic sea ice drift estimation merging radiometer and scatterometer data. *IEEE Transactions on Geoscience and Remote Sensing*, 50(7):2639–2648, 2012. doi:10.1109/TGRS.2012.2184124.

[7] J. Haarpaintner. Arctic-wide operational sea ice drift from enhanced-resolution Quikscat/SeaWinds scatterometry and its validation. *IEEE Transactions on Geoscience and Remote Sensing*, 44(1):102–107, January 2006.

[8] T. W. Haine. Arctic freshwater export: status, mechanisms, and prospects. *Global Planet. Change*, 125:13–35, 2015.

[9] P. Holland and R Kwok. Wind-driven trends in antarctic sea-ice drift. *Nature Geosciencies*, 2012. doi:https://doi.org/10.1038/ngeo1627.

[10] Alexander S. Komarov and David G. Barber. Sea ice motion tracking from sequential dual-polarization radarsat-2 images. *IEEE Transactions on Geoscience and Remote Sensing*, 52(1):121–136, 2014. doi:10.1109/TGRS.2012.2236845.

[11] T. Krumpen, H.J. Belter, and A. et al. Boetius. Arctic warming interrupts the transpolar drift and affects long-range transport of sea ice and ice-rafted matter. *Sci Rep*, 2019. doi:https://doi.org/10.1038/s41598-019-41456-y.

[12] R. Kwok. Summer sea ice motion from the 18 GHz channel of AMSR-E and the exchange of sea ice between the Pacific and Atlantic sectors. *Geophysical Research Letters*, 2008. doi:10.1029/2007GL032692.

[13] R. Kwok, J. C. Curlander, R. McConnel, and S. Pang. An ice-motion tracking system at the Alaska SAR Facility. *IEEE Journal of Oceanic Engineering*, 15:44–54, 1990.

[14] R. Kwok, A. Schweiger, D. A. Rothrock, S. Pang, and C. Kottmeier. Sea ice motion from satellite passive microwave imagery assessed with ERS SAR and buoy motions. *Journal of Geophysical Research*, 103:8191–8214, April 1998. doi:10.1029/97JC03334.

[15] R. Kwok, G. Spreen, and S. Pang. Arctic sea ice circulation and drift speed: decadal trends and ocean currents. *Journal of Geophysical Research: Oceans*, 118(5):2408–2425, 2013. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/jgrc.20191, arXiv:https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1002/jgrc.20191, doi:https://doi.org/10.1002/jgrc.20191.

[16] Ron Kwok, Shirley S. Pang, and Sahra Kacimi. Sea ice drift in the Southern Ocean: Regional patterns, variability, and trends. *Elementa: Science of the Anthropocene*, 06 2017. 32. doi:10.1525/elementa.226.

[17] Jeffrey C. Lagarias, James A. Reeds, Margaret H. Wright, and Paul E. Wright. Convergence properties of the Nelder-Mead simplex method in low dimensions. *SIAM Journal on Optimization*, 9:112–147, 1998.

[18] T. Lavergne, M. Piñol Solé, E. Down, and C. Donlon. Towards a swath-to-swath sea-ice drift product for the copernicus imaging microwave radiometer mission. *The Cryosphere*, 15(8):3681–3698, 2021. URL: https://tc.copernicus.org/articles/15/3681/2021/, doi:10.5194/tc-15-3681-2021.

[19] Thomas Lavergne, Steinar Eastwood, Zakaria Teffah, Harald Schyberg, and Lars-Anders Breivik. Sea ice motion from low resolution satellite sensors: an alternative method and its validation in the Arctic. *Journal of Geophysical Research*, 2010. doi:10.1029/2009JC005958.

[20] M. Leppäranta. *The drift of sea ice*. Springer-Praxis Books in Geophysical Sciences. Springer-Verlag, 2nd edition, 2011.

[21] Anthony K. Liu, Yunhe Zhao, and Sunny Y. Wu. Arctic sea ice drift from wavelet analysis of NSCAT and Special Sensor Microwave Imager data. *Journal of Geophysical Research*, 104(C5):11529–11538, 1999.

[22] J. Maslanik, T. Agnew, M. Drinkwater, W. Emery, C. Fowler, R. Kwok, and A. Liu. Summary of ice-motion mapping using passive microwave data. Technical Report Special Publication 8, NSIDC — National Snow and Ice Data Center, November 1998. URL: \{http://nsidc.org/pubs/special/nsidc_special_report_8.pdf\}.

[23] S. Muckenhuber, A. A. Korosov, and S. Sandven. Open-source feature-tracking algorithm for sea ice \hack \newline drift retrieval from sentinel-1 sar imagery. *The Cryosphere*, 10(2):913–925, 2016. URL: https://tc.copernicus.org/articles/10/913/2016/, doi:10.5194/tc-10-913-2016.

[24] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computational Journal*, 7:308–313, 1968.

[25] R. M. Ninnis, W. J. Emery, and M. J. Collins. Automated extraction of pack ice motion from advanced very high-resolution radiometry. *Journal of Geophysical Research*, 91:10725–10734, 1986. doi:10.1029/JC091iC09p10725.

[26] Giulio Notarstefano, Pierre-Marie Poulain, and Elena Mauri. Estimation of surface currents in the Adriatic sea from sequential infrared satellite images. *Journal of Atmospheric and Oceanic Technology*, 25:271–285, May 2007. doi:10.1175/2007JTECHO527.1.

[27] K.I. Ohshima, S. Nihashi, and K. Iwamoto. Global view of sea-ice production in polynyas and its linkage to dense/bottom water formation. *Geosci. Lett. 3, 13*, 2016. doi:https://doi.org/10.1186/s40562-016-0045-4.

[28] Johannes Schmetz, Kenneth Holmlund, Joel Hoffman, Bernard Strauss, Brian Mason, Volker Gærtner, Arno Koch, and Leo van de Berg. Operational cloud-motion winds from Meteosat infrared images. *Journal of Applied Meteorology*, 32:1206–1225, July 1993.

[29] Gunnar Spreen, Ron Kwok, and Dimitris Menemenlis. Trends in arctic sea ice drift and role of wind forcing: 1992–2009. *Geophysical Research Letters*, 38(19):, 2011. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2011GL048970, arXiv:https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2011GL048970, doi:https://doi.org/10.1029/2011GL048970.

[30] Mary-Louise Timmermans and John Marshall. Understanding arctic ocean circulation: a review of ocean dynamics in a changing climate. *Journal of Geophysical Research: Oceans*, 125(4):e2018JC014378, 2020. e2018JC014378 10.1029/2018JC014378. URL: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2018JC014378, arXiv:https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/2018JC014378, doi:https://doi.org/10.1029/2018JC014378.

INDEX