

PyTroll

An Open-Source, Community-Driven Python Framework to Process Earth Observation Satellite Data

MARTIN RASPAUD, DAVID HOESE, ADAM DYBBROE, PANU LAHTINEN, ABHAY DEVASTHALE, MIKHAIL ITKIN, ULRICH HAMANN, LARS ØRUM RASMUSSEN, ESSEN STIGÅRD NIELSEN, THOMAS LEPELT, ALEXANDER MAUL, CHRISTIAN KLICHE, AND HROBJARTUR THORSTEINSSON

Over the last few decades, there has been a rapid increase in both the amount and diversity of Earth Observation (EO) satellite data. Much of these data are assimilated to improve numerical weather prediction (NWP) (Rabier 2005), while a significant portion are tailored for nowcasting and early warning. Today's society and global economy is more vulnerable than yesterday's to weather and environmental disasters (e.g., floods, forest fires, desert dust, volcanic eruptions). Consequently, there is a growing public demand for early and accurate forecasts and warnings to be able to react quickly and minimize damage to the economy and society.

Real-time 24/7 satellite processing systems like those being operated at the national weather services are being challenged by a steady increase in the amount

of satellite data and ever-evolving user requirements. The users are more diversified than ever and include forecasters, decision-makers, and other end users.

To achieve a durable satellite processing system supporting highly specialized nowcasting products, we argue the need for an open and resilient software platform that is able to both quickly adapt to new data sources and respond efficiently to new and often unexpected user needs. Furthermore, open-source development stimulates international collaboration, which is a necessity given that there are often rather limited resources on the national level to operate and maintain a 24/7 production system under the very demanding conditions outlined above.

This has been the motivation for the development of the Python-based open software framework, *PyTroll* (<http://pytroll.org>), originally started in 2009 as a collaboration between the Danish Meteorological Institute (DMI) and the Swedish Meteorological and Hydrological Institute (SMHI).¹ Python was the programming language of choice for several reasons. Python is a widely used, free, and open-source interactive programming language. It allows for a flexible bottom-up and fast software development cycle, and both SMHI and DMI have had a long tradition of using Python in operational remote sensing. It is now more common to see Python being used as an end-to-end solution with today's comprehensive standard library and mature support for optimized numerical and scientific computations (Lin 2012). With tools like Cython, Numexpr, and PyPy, a system built purely upon the Python ecosystem often competes well in performance with solutions based on compiled languages, while preserving its flexibility and ease of use.

AFFILIATIONS: RASPAUD, DYBBROE, AND DEVASTHALE—Swedish Meteorological and Hydrological Institute, Norrköping, Sweden; HOESE—Space Science and Engineering Center (SSEC), University of Wisconsin—Madison, Madison, Wisconsin; LAHTINEN—Finnish Meteorological Institute (FMI), Helsinki, Finland; ITKIN—Norwegian Polar Institute (NPI), Tromsø, Norway; HAMANN—Federal Office of Meteorology and Climatology (MeteoSwiss), Locarno-Monti, Switzerland; RASMUSSEN—Danish Meteorological Institute (DMI), Copenhagen, Denmark; NIELSEN—Think Big Analytics, Copenhagen, Denmark; LEPELT AND MAUL—Deutscher Wetterdienst (DWD), Offenbach am Main, Germany; KLICHE—EBP Deutschland GmbH, Berlin, Germany; THORSTEINSSON—Videntifier Technologies ehf, Reykjavik, Iceland

CORRESPONDING AUTHOR: Martin Raspaud, martin.raspaud@smhi.se

DOI:10.1175/BAMS-D-17-0277.1

A supplement to this article is available online (10.1175/BAMS-D-17-0277.2).

©2018 American Meteorological Society

For information regarding reuse of this content and general copyright information, consult the [AMS Copyright Policy](#).

¹ The historical evolution of the *PyTroll* code base can be seen in this video: <https://youtu.be/tZcKpB6yPW0>.

PyTroll, being open source and freely available on the web from its beginning, has fostered international cooperation so that today it encompasses a truly worldwide community. This collaboration is stimulated via biannual *PyTroll* 5-day code sprints, where both developers and users contribute. *PyTroll* is currently being used operationally in about one-third of the EUMETSAT member states, including the national meteorological services of Denmark, Norway, Sweden, Finland, Germany, Switzerland, Italy, Estonia, and Latvia. While *PyTroll* was originally developed to cater to the needs of the atmospheric remote sensing community, it is equally suitable for land, ocean, or hydrology applications of EO data. *PyTroll* supports processing of a great number of current and past EO satellites, including the recent Copernicus Sentinel programs of the European Space Agency (ESA), NOAA's *GOES-16*, the joint NOAA–NASA *JPSS-1* (NOAA-20), and JMA's *Himawari 8* and 9.

By providing small stand-alone packages, *PyTroll* attempts to provide as much freedom to the user as possible while still providing a large array of available features. Users are then free to develop their own customized systems. For example, the Community Satellite Processing Package (CSPP) Polar2Grid project (www.ssec.wisc.edu/software/polar2grid/), created by the Space Science and Engineering Center (SSEC) at the University of Wisconsin, provides a simple command line interface to the features provided by *PyTroll* packages in a free and open-source all-in-one installation. By doing this, Polar2Grid lets users unfamiliar with programming, Python environments, or certain satellite image processing techniques produce atmospherically corrected, high-resolution, high-contrast imagery.

A summary overview of all *PyTroll* packages, their interdependencies, level of maturity, and known operational usage is listed on the *PyTroll* home page (http://pytroll.github.io/pytroll_packages_overview.html). Here the readers also find various code examples using *PyTroll* (<https://nbviewer.jupyter.org/github/pytroll/pytroll-examples/tree/master/>). In the following sections we highlight the main capabilities of *PyTroll* packages. We begin by describing the *SatPy* package that is central to *PyTroll* and most frequently used, followed by describing packages that support satellite data acquisition and sharing in real time in the most efficient manner. A suite of utilities available to facilitate 24/7 data production from reception to Level 2 product generation are presented in the next section. Finally, we introduce a package

that allows derivation of fundamental climate data record from the historical sensors. Prospective *PyTroll* users are welcome to ask questions on Slack (<https://pytroll.slack.com/>); get an invitation here: <https://pytrollslackin.herokuapp.com/>) or send an e-mail to the *PyTroll* mailing list (<https://groups.google.com/group/pytroll>). The potential contributors can download the *PyTroll* code on Github, and after adding their own features of interest or modifying the existing ones, they can make issues or pull requests as needed.

LOAD, PROCESS, AND DISPLAY. A rather common requirement in Earth sciences is to read radiance data; perform calibration, geolocation, and resampling; and then generate an image for display. All of this is handled in a unified manner in *PyTroll* via the high-level package *SatPy*. *SatPy* currently supports a wealth of different input formats (<http://satpy.readthedocs.io/en/latest/>) for various EO satellite instruments and advanced processing algorithms.

The *SatPy* package is easily installed in existing Python environments with pip install satpy on the command-line. *SatPy* allows users to perform the data processing without any additional configuration. Methods on how to read input files, generate RGB image composites, and other necessary processing are all provided in the configuration that comes with the package. *SatPy* makes it easy for complex correction or composite algorithms to work across multiple sensors by providing a flexible set of metadata for each dataset. Metadata includes dataset name, wavelength, calibration, satellite name, and observation start and end times, among other information. Algorithms can be configured to request datasets by wavelength, rather than by name, which allows one specification to work for multiple sensors.

To generate an image, as shown in Fig. 1, it is necessary to calibrate the raw data to reflectances, normalize illumination using solar zenith angle, and reduce the effect of Rayleigh scattering from the visual bands. *SatPy* is able to do all this with the help of other *PyTroll* packages like *PySpectral* and *PyOrbital*, which are automatically installed with the aforementioned pip command. The data may be resampled to a cartographic map projection and displayed as an image or stored to a file on disk. *SatPy* supports writing to Network Common Data Form Climate and Forecast (netCDF/CF) and various image formats, including GeoTIFF and PNG. The left column of Fig. 1 illustrates a typical *SatPy* use case of generating a so-called true-color RGB composite,

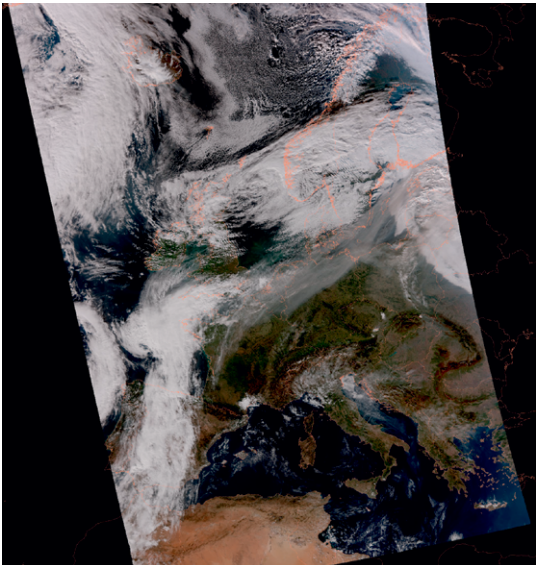
<pre>from glob import glob from satpy.scene import Scene scn = Scene(filename=glob(...), reader='viirs_sdr') scn.load(['true_color']) newscn = scn.resample('eurol') newscn.show('true_color', overlay={'coast_dir': '/data/shapes/', 'color': (255, 140, 100)})</pre>	<pre>from satpy.scene import Scene scn = Scene(base_dir='/data/seviri/hrit/', reader='hrit_msg') scn.load([0.6, 0.8]) # wavelength in um newscn = scn.resample('eurol') newscn[0.6]</pre>
 On October 17, 2017, an air mass with a heavy load of particles and gases stemming from extensive and devastating forest/grass fires on the Iberian Peninsula reached Northern Europe. Probably the same air mass also carried a significant amounts of Saharan dust from dust outbreaks from Western Africa a few days earlier. The dirty air mass is clearly seen with the Suomi-NPP VIIRS sensor after correcting for the atmospheric background signal using <i>PySpectral</i>.	<pre><xarray.DataArray 'astype-e3d8a3' (y: 3712, x: 3712)> dask.array<concatenate, shape=(3712, 3712), dtype=float64, chunksize=(53, 3712)></pre> Coordinates: <pre>* x (x) float64 -5.569e+06 ... * y (y) float64 5.566e+06 ...</pre> Attributes: <pre>units: % wavelength: (0.56, 0.635, 0.71) standard_name: toa_bidirectional_reflectance platform_name: Meteosat-10 sensor: seviri satellite_longitude: 0.0 satellite_latitude: 0.0 satellite_altitude: 35785831.0 start_time: 2017-01-19 09:30:10.553000 end_time: 2017-01-19 09:42:41.403000 area: Area ID: Ontheflyarea name: VIS006 resolution: 3000.40316582 calibration: reflectance polarization: None modifiers: () ancillary_variables: []</pre>

FIG. 1. (left) The code to generate a ratio-sharpened true-color image with an overlay of coastlines and political borders. The sharpening is achieved by multiplying the low-resolution green and blue channels by the ratio of the high-resolution red channel and its lower-resolution counterpart (Gumley et al. 2003). This ratio-sharpening technique is available in *SatPy* and can be used for any sensor with both low- and high-resolution bands. This code example is available as a Jupyter notebook document on Github: https://github.com/pytroll/pytroll-examples/blob/master/satpy/viirs_smoke.ipynb. The example code above took 105 s to execute on a laptop running RedHat Enterprise Linux version 7, with a dual core hyperthreaded processor [Intel(R) Core(TM) i7-4600U CPU @ 2.10GHz] and 16 GB RAM (DDR3 1600Mhz) and using local SSD storage (256GB SanDisk SD7TB3Q). The ratio-sharpened true color requires more memory and is slower to run due to the higher-resolution data used. The low-resolution true color using only the M-bands take 43 s to generate on the same machine. **(right)** The code to load two Meteosat SEVIRI channels based on their wavelength in micrometers.

resampling it, and displaying it on-screen. A true-color image is obtained by combining a red, a green, and a blue visible wavelength band, for example, with the Visible and Infrared Imaging Radiometer Suite (VIIRS) instrument on board the Suomi NPP and JPSS satellites, that would be, respectively, the M4 (or I1), M3, and M2 bands. The right column of Fig. 1 illustrates another use case with loading two

Spinning Enhanced Visible and Infrared Imager (SEVIRI) channels by wavelength.

Once data have been loaded, calibrated, georeferenced, and atmospherically corrected, the data are resampled from the native satellite projection to a custom grid. This is illustrated in Python code in Fig. 1 with `newscn = scn.resample('eurol')`. All *SatPy* resampling functionality comes from the

PyResample package, which provides a fast generalized interface for resampling geolocated data. The *PyTroll* Python-geotiepoints package allows interpolation of geolocation information in the case when the level 1 data only contain geolocation on a grid of tie points. *PyResample* utilizes a custom k -dimensional (k -d) tree algorithm implementation provided by the *PyKDTree* package. The k -d tree nearest neighbor search algorithm (Bentley 1975) implemented in *PyKDTree* is highly efficient due to its use of Cython and OpenMP, and it is faster than the Scipy and libann (www.cs.umd.edu/~mount/ANN/) packages. Other resampling methods included in *PyResample* are the Bilinear Interpolation and the Elliptical Weighted Average methods. Finally, adding coastlines and political borders to the image can be done using the *PyCoast* package. By default, *PyCoast* uses vector data from the Global Self-consistent, Hierarchical, High-resolution Geography (GSHHG) Database provided by the National Oceanic and Atmospheric Administration (NOAA).

SHARING DIRECT READ-OUT DATA IN REAL TIME.

Prior to reading, the satellite data needs to be retrieved and stored locally. *PyTroll* provides dedicated tools supporting data acquisition, namely *PyTroll-Schedule*, *Trollcast*, *Posttroll*, and *Trollmoves*.

The *PyTroll-Schedule* package provides a means of computing an optimized reception schedule for polar-orbiting satellite overpasses seen from single or multiple direct-readout stations. Given a single-station position, an area of interest, a list of satellites, and a weight assigned to each satellite, the scheduler process computes all the possible overpasses for the satellites over the station before choosing the sequence of overpasses that optimizes the coverage over the area of interest (examples are shown in Fig. 2 and in the online supplementary figure at <https://doi.org/10.1175/BAMS-D-17-0277.2>). *PyTroll-Schedule* also has the capability to plan the reception schedules of several antennas working together. Making the assumption that all data received at one

single antenna station is available to the others, we take advantage of the multiple stations to optimally schedule the reception of as many satellites as possible. *PyTroll-Schedule* needs information on the position of the satellites relative to an observer on the ground (given by the location of the antenna). This can be determined with the use of *PyOrbital*. It allows computation of the position of a satellite in space at a given time through the use of Two-Line Element (TLE) sets and the SGP4 algorithm that has been implemented in Python from the original work of Vallado et al. (2006).

Exchanging polar satellite data in real time from several antennas connected over the Internet is a challenge that *PyTroll* tries to address with the *Trollcast* package. *Trollcast* is loosely based on the concept of the BitTorrent protocol (Cohen 2008), with dissemination of small packages of data upon request. This means the raw satellite data are split into small chunks—typically a single scanline or a unit packet—that can be uniquely identified. These

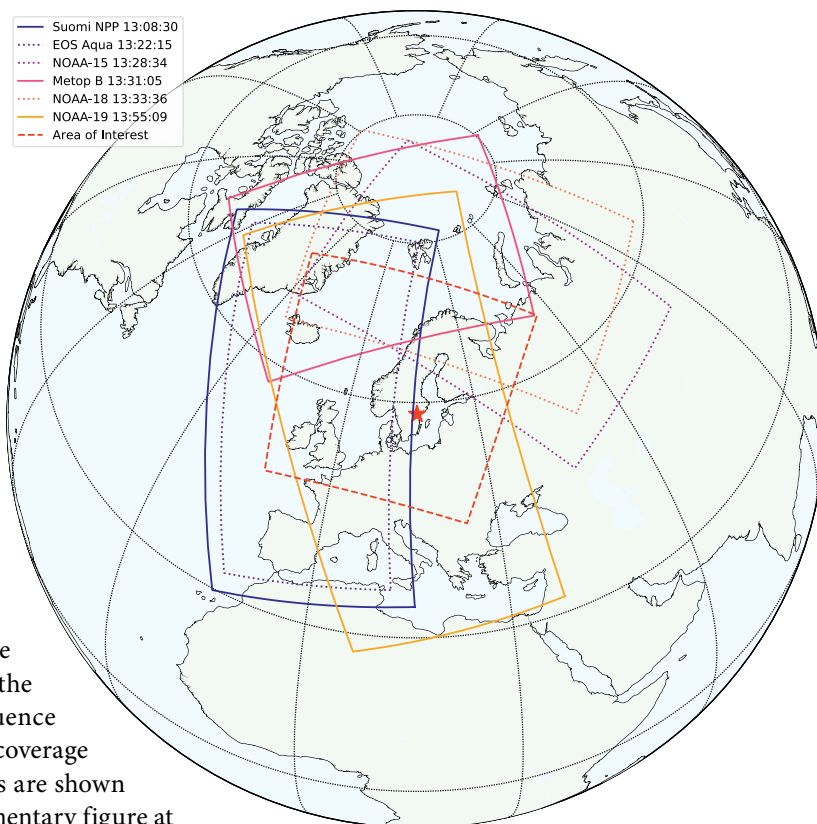


FIG. 2. One hour of passes, covering the interest area depicted in red dashes. Solid lines are the passes chosen for reception. The red star indicates the position of the reception station.

chunks are then advertised to the potential receivers of the data. The receivers can make requests to the interesting data source(s) to reconstruct the entire raw satellite dataset to fit their needs locally. A quality indicator, and other metadata associated with the data, is published along with the advertisement of the data chunk. By exploiting this information, the receiver is able to combine data from multiple sources and optimize the data quality locally.

This data distribution scheme is implemented in the national meteorological services (NMSs) of Norway, Finland, and Sweden. It efficiently connects these neighboring but geographically separated antennas, improving local reception quality and increasing the number of available overpasses at each NMS. At these high latitudes, for a reception taking in data from eight satellites, a single antenna can only receive between 50% and 60% of the available overpasses, while three stations coordinating their schedules could receive 100% of the overpasses.

Another omnipresent task for the satellite-reception station operator is shuffling information and data around. For this, two packages are available in *PyTroll*: *Posttroll* and *Trollmoves*. *Posttroll* is a communication library built around the very efficient and powerful ZeroMQ (Hintjens 2013). It currently implements the publisher–subscriber and request–reply communication paradigms, along with help function to encode and decode standard *PyTroll* messages. A *PyTroll* message might, for example, contain information that a satellite scene has been processed and a file is available on disk. While *Posttroll* was developed with the purpose of connecting and exchanging information between individual processing tasks in a fully automated real-time *PyTroll* batch processing system (see below), its generic design may allow for other user applications. As for data, *Trollmoves* provides a utility for moving files from one place to another on a computer network. It uses a reliable client–server architecture where the server advertises available data and the client makes requests for relevant data.

24/7 PRODUCTION. The next logical step is to start product generation from the received data. Here, *PyTroll* offers tools for a lightweight generic and distributed workflow framework. *Trollflow* allows encapsulating smaller processing steps (like reading level 1 data and generating composites), while being connected to other processing steps through messages using *Posttroll*. Two other *PyTroll* tools, *PyTroll-Runners* and *PyTroll-Collectors*, allow the running of

third-party software within the workflow and the collection of segmented data, respectively.

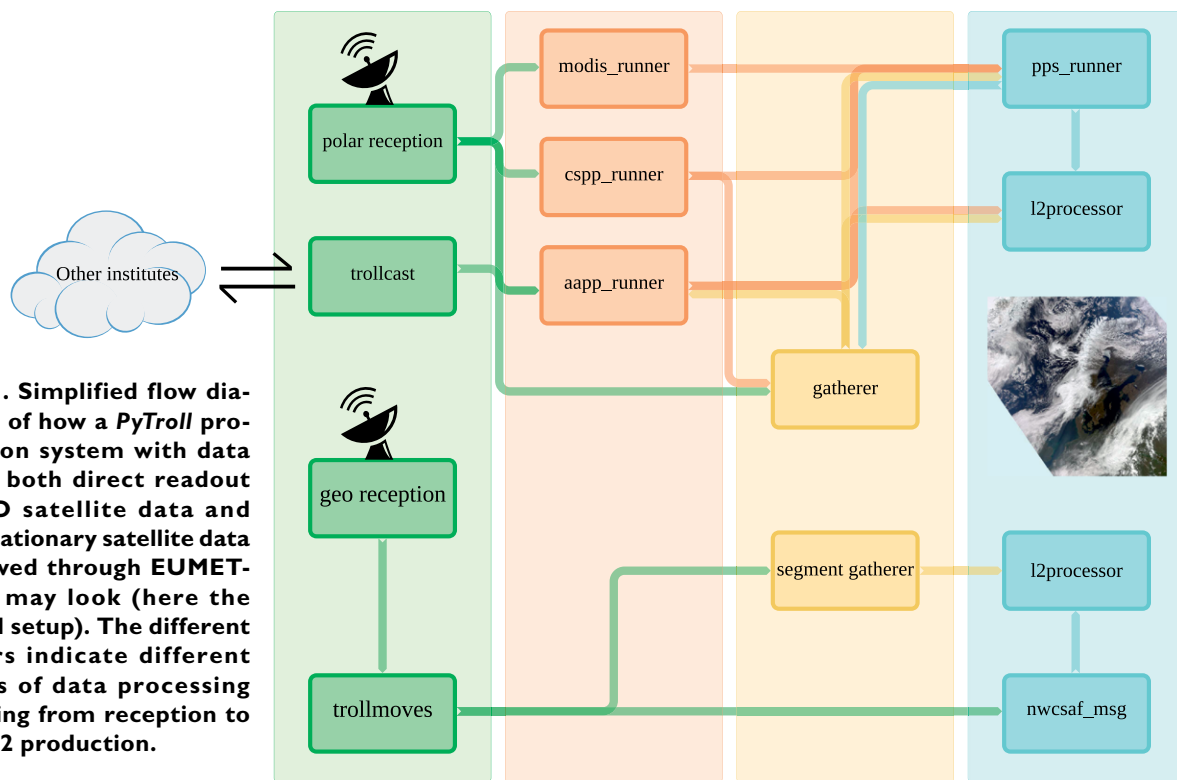
Logical processing steps can be determined to go from the reception of raw data to a final product that can be typically ingested into a visualization system at the forecasters desk or used in a data-assimilation system for NWP. Such a sequence of steps can be:

- 1) Moving data files to the production server
- 2) Filtering or gathering of data granules
- 3) Geolocating and calibrating the raw data (level 0) to radiances, brightness temperatures, and reflectances (level 1)
- 4) Processing the data, making RGB image composites, or the derivation of level 2 parameters, including resampling to a predefined grid
- 5) Writing and storing the data to disk

To demonstrate this concept, a part of Fig. 3 shows the current processing flow at SMHI for the direct readout of polar satellite data. Data from Advanced Very High Resolution Radiometer (AVHRR), Moderate resolution Imaging Spectroradiometer (MODIS), and VIIRS on board the NOAA/Metop, *EOS-Terra*, *EOS-Aqua*, and *Suomi National Polar-Orbiting Partnership (Suomi-NPP)* satellites are being processed in real time with *PyTroll*. Data are received via the antenna in Norrköping, Sweden, or from nearby antennas in Norway and Finland via *Trollcast* (green in Fig. 3). Three level 1 runners (darker orange in Fig. 3), one for each instrument, encapsulating three different third-party processing packages (AAPP, CSPP, and SeaDAS), manage the processing of the raw data records (RDR/level-0) to Scientific Data Records (SDR/level-1). The VIIRS SDR granules produced by CSPP with the *cspp_runner* are being collected by the *gatherer* (lighter orange in Fig. 3), so that when enough available granules are covering the local area of interest a message is published, and the processing of RGB imagery and level 2 products can start (blue in Fig. 3). The *flow-processor*, as provided in *Trollflow*, encapsulates *SatPy* and will produce a number of rectified image products, typically in GeoTIFF² and PNG formats, from level 1 data from all sensors. The *ppls_runner* encapsulates another third-party package, the NWCSAF/PPS (Dybbroe et al. 2005), that

² *PyTroll* supports a set of special Tiff tags (NinjoTiff) applicable to the Meteorological Workstation NinJo, used in several countries in Europe and in Canada (www.ninjo-workstation.com/).

FIG. 3. Simplified flow diagram of how a PyTroll production system with data from both direct readout of EO satellite data and geostationary satellite data received through EUMET-Cast may look (here the SMHI setup). The different colors indicate different levels of data processing starting from reception to level-2 production.



produces cloud parameters as netCDF files. Some image products require both the level 1 radiances and the level 2 cloud parameters. In that case, the *gatherer* will wait until all input data are available before messaging the *flow processor*.

Trollflow is used in operations at the Finnish Meteorological Institute for MSG (0° service, RSS and IODC), *Himawari-8* and *GOES-15* geostationary satellites, and AVHRR/3 (both Metop and NOAA, direct broadcast and EARS) and *Suomi-NPP*/VIIRS (EARS and direct broadcast) polar satellites. As an example, the MSG 0° service is processed for five different area definitions for the European area, and altogether 38 different composites are created. Typically, the processing for this set takes 43 s for a daylight scene, and less than 25 s for nighttime scenes when there are less valid composites. The machine used in this case is a Red Hat Enterprise Linux virtual machine with 4 GB RAM and one CPU core (Intel Xeon) running at 2.40GHz.

PYTROLL FOR FACILITATING THE DERIVATION OF CLIMATE DATA RECORDS.

While the operational needs triggered the original development of *PyTroll* packages, it offers possibilities to process EO data to cater climate research needs,

as well. For example, the *PyGAC* package, developed in the framework of ESA's Cloud Climate Change Initiative (ESA Cloud CCI, www.esa-cloud-cci.org/), allows reading, decoding, and calibration of 4-km Global Area Coverage (GAC) data from the AVHRR instruments on board a series of NOAA and Metop satellites (Devasthale et al. 2017). These 34 years of data, available from 1978 to present, have been recently processed using *PyGAC* in the frameworks of ESA Cloud CCI and EUMETSAT's Satellite Application Facility for Climate Monitoring (CMSAF, www.cmsaf.eu) projects (Schulz et al. 2009; Hollmann et al. 2013). These data were the basis for the subsequent retrievals of the Thematic Climate Data Records (TCDR) of global cloud properties (Karlsson et al. 2017; Stengel et al. 2017). As a demonstration, Fig. 4 shows the climatological-mean reflectance derived from AVHRR channel 1 (0.65 μm) using *PyGAC* for the 34-yr period from 1982 to 2015. In addition to 4-km GAC data, *PyGAC* is capable of reading, decoding, and calibrating 1-km Local Area Coverage (LAC) data from AVHRR for climate applications. For example, Pareeth et al. (2016) used *PyGAC* to derive geometrically corrected historical time series of brightness temperatures based on 28 years of 1-km LAC data (1986–2014) over the alpine lakes of northern Italy.

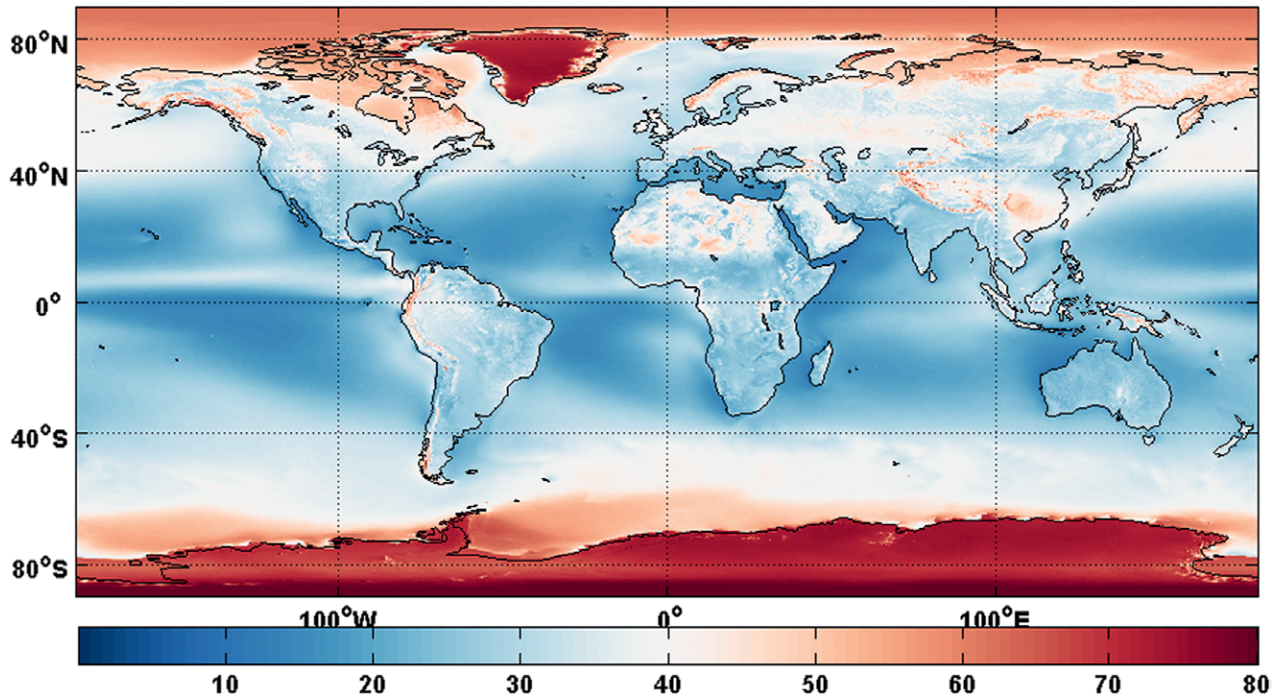


FIG. 4. Annual-mean climatological reflectance at $0.65\ \mu\text{m}$ for the 34-yr period (1982–2015) based on daytime AVHRR data from the afternoon satellites.

Users interested in investigating individual GAC and/or LAC orbits can use *SatPy* for easy reading, manipulation, and visualization, using the same method as presented in Fig. 1.

FUTURE WORK. By February 2017, a handful of the European NMSs had signed a Memorandum of Understanding (http://pytroll.github.io/pytroll_mou_20170222.pdf) showing their long-term commitment to develop and maintain the *PyTroll* software for operational use. All *PyTroll* packages are available on GitHub (<https://github.com/pytroll>) for sustainability, which integrates with various free cloud services that allows continuous automatic code testing ([Travis-ci.com](https://travis-ci.com) and Appveyor); promotes good code documentation ([ReadTheDocs.io](https://readthedocs.io)), code standards ([Landscape.io](https://landscape.io) and Codacy), and test coverage ([Coveralls.io](https://coveralls.io)); and thus secures an overall healthy and robust code base. Thus far, *PyTroll* has been mainly focusing on imager data, but activities are ongoing to better integrate other data sources, like the *GOES-16* Lightning Mapper data and Hyperspectral sounder data like the Infrared Atmospheric Sounding Interferometer (IASI) on Metop and Cross-Track Infrared Sounder (CrIS) on *Suomi-NPP* and JPSS platforms. Work is progressing to enhance *Trollcast* to support the

channel access data unit (CADU) file format in addition to high-resolution picture transmission (HRPT) format, which will allow sharing data from all current polar EO satellites. Recent improvements have been made to *SatPy* to more effectively handle large data arrays both on single workstations and data clusters, which is becoming increasingly important with the recent launches of *Himawari-8* and *Himawari-9* and *GOES-16*, and with new satellite missions on the horizon, such as Meteosat Third Generation (MTG) and Metop Second Generation (EPS-SG) polar satellites. To handle the very high data rates from these more recent and future missions in a real-time batch processing system, work toward distributed processing is being pursued (e.g., with the Dask library).

FOR FURTHER READING

- Bentley, J. L., 1975: Multidimensional binary search trees used for associative searching. *Commun. ACM*, **18**, 509, <https://doi.org/10.1145/361002.361007>.
- Cohen, B., 2008: The BitTorrent protocol specification. www.bittorrent.org/beps/bep_0003.html.
- Devasthale, A., and Coauthors, 2017: *PyGAC*: An open-source, community-driven Python interface to preprocess nearly 40-year AVHRR global area

- coverage (GAC) data record. *GSICS Quarterly Newsletter*, **11**, 3–5.
- Dybbroe, A., A. Thoss, and K.-G. Karlsson, 2005: NWCSAF AVHRR cloud detection and analysis using dynamic thresholds and radiative transfer modeling. Part 1: Algorithm description. *J. Appl. Meteor.*, **44**, 39–54, <https://doi.org/10.1175/JAM-2188.1>.
- Gumley, L., J. Descloîtres, and J. Schmaltz, 2003: Creating reprojected true color MODIS images: A tutorial.
- Hintjens, P., 2013: *ZeroMQ: Messaging for Many Applications*. O'Reilly Media, https://books.google.se/books?id=TxHgtl_sFmgC.
- Hollmann, R., and Coauthors, 2013: The ESA climate change initiative: Satellite data records for essential climate variables. *Bull. Amer. Meteor. Soc.*, **94**, 1541–1552, <https://doi.org/10.1175/BAMS-D-11-00254.1>.
- Karlsson, K.-G., and Coauthors, 2017: CLARA-A2: The second edition of the CM SAF cloud and radiation data record from 34 years of global AVHRR data. *Atmos. Chem. Phys.*, **17**, 5809–5828.
- Lin, J. W.-B., 2012: Why Python is the next wave in Earth sciences computing. *Bull. Amer. Meteor. Soc.*, **93**, 1823–1824, <https://doi.org/10.1175/BAMS-D-12-00148.1>.
- Pareeth, S., and Coauthors, 2016: New automated method to develop geometrically corrected time series of brightness temperatures from historical AVHRR LAC data. *Remote Sens.*, **8**, <https://doi.org/10.3390/rs8030169>.
- Rabier, F., 2005: Overview of global data assimilation developments in numerical weather-prediction centres. *Quart. J. Roy. Meteor. Soc.*, **131**, 3215–3233, <https://doi.org/10.1256/qj.05.129>.
- Schulz, J., and Coauthors, 2009: Operational climate monitoring from space: The EUMETSAT Satellite Application Facility on Climate Monitoring (CM-SAF). *Atmos. Chem. Phys.*, **9**, 1687–1709, <https://doi.org/10.5194/acp-9-1687-2009>.
- Stengel, M., and Coauthors, 2017: Cloud property datasets retrieved from AVHRR, MODIS, AATSR and MERIS in the framework of the Cloud_cci project. *Earth Syst. Sci. Data*, **9**, 881–904, <https://doi.org/10.5194/essd-9-881-2017>.
- Vallado, D. A., P. Crawford, R. Hujsak, and T. S. Kelso, 2006: Revisiting Spacetrack Report #3. *AIAA Astro-dynamics Specialists Conf. and Exhibit*, Keystone, CO.

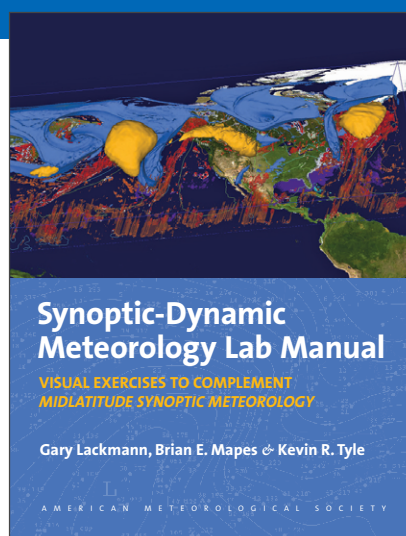
NEW FROM AMS BOOKS!

Synoptic-Dynamic Meteorology Lab Manual Visual Exercises to Complement Midlatitude Synoptic Meteorology

This lab manual represents a systematic, applied approach to meteorological education that allows well-established theoretical concepts to be used with modernized observational and numerical datasets. Its groundbreaking visualizations elucidate concepts taught in the companion textbook by Gary Lackmann, *Midlatitude Synoptic Meteorology*.

Gary M. Lackmann, North Carolina State University
Brian E. Mapes, University of Miami
Kevin R. Tyle, University at Albany, State University of New York

© 2017, paperback and ebook, 126 pages
 ISBN 13: 978-1-878220-26-4
 eISBN: 978-1-935704-65-2
 List price: \$80
 AMS Member price: \$60 Student price: \$50



AMS BOOKS
 RESEARCH APPLICATIONS HISTORY
bookstore.ametsoc.org