AUTOMATED SPECTRAL IMAGE PROCESSING TECHNIQUES IN THE MARSLAB FAMILY OF APPLICATIONS. M. A. St. Clair,[1] C. C. Million,[1] S. Brown,[1] and M.S. Rice.[2]
[1]Million Concepts (mstclair@millionconcepts.com), [2]Western Washington University.

Introduction: *marslab* is a suite of interoperable but loosely-coupled libraries and applications that facilitate rapid production of robust, stable, and flexible data analysis pipelines and workflows. It is principally designed for, but not restricted to, analysis of remote sensing multispectral data. It is in tactical use on the Mars 2020 mission for Mastcam-Z (ZCAM) multispectral operations and has also been extensively applied to MSL Mastcam (MCAM) data. [e.g. 1,2] We have previously described its user-facing functionality in [3]. Here, we discuss implementation details of some of its image processing structures. Many are fundamentally general-purpose; most others could be easily modified for use with any number of spectral datasets. Its frameworks are specifically designed for flexible definition of instrument-specific behavior.

Availability. All *marslab* code is released under the permissive open-source BSD 3-Clause license. See [4] for summary descriptions of individual libraries and links to their GitHub repositories. Some portions of the code (primarily from the *asdf* handler application) are open-source but non-public due to mission confidentiality requirements.

Languages and frameworks (non-exhaustive). All *marslab* libraries are primarily written in Python. Applications in the suite that run in-browser (VISOR & MultiDEx) also use JS for some GUI elements. All applications lean heavily on 'standard' scientific Python libraries (*numpy, pandas, scipy*, etc.). Many *marslab* image processing operations rely on fundamental algorithms from *scikit-image* and *opencv*, and it performs most of its 'headless' (non-GUI) rendering operations using *matplotlib* and *pillow.* In general, *marslab* attempts to pass as much of its computationally hard work as possible to the highly-optimized C and Fortran extensions that back the scientific Python ecosystem. Some parts of the suite rely on cloud APIs (specifically, AWS Lambda, S3, and EC2; and Google Drive and Sheets), but we do not specifically discuss those components here.

Browse Product Pipelines: Multispectral data have many possible human-readable interpretations; analysts require access to a diversity of 'browse' views on these data to understand them. To help guide tactical decisions, it is crucial to generate browse products *quickly* and *consistently*. However, 'correct' standards for browse products often change during an investigation (due to evolving interpretations, decaying hardware, etc.); furthermore, rapid implementation of new products can often address emergent tactical needs (e.g., visualizing a different spectral parameter might help analysts select a target for proximity science). However, due to the complexity of multispectral image processing pipelines, consistency and responsivity are often competing priorities.

`Composition and Look`. To help resolve this conflict, we have created a general-purpose framework for domain-specific functional programming in Python (`Composition`), along with an implementation of that framework intended for easy definition of image processing pipelines (`Look`). This system is designed for extensibility to a wide variety of instruments and data sources; along with the production MCAM and ZCAM implementations, reference implementations exist for CH-1 M3 hyperspectral images and 'consumer' RGB(A) image formats. `Look` abstracts messy data flow and API glue by 'compiling' pipeline descriptions written in a markup-like subset of Python into `Compositions` of partially-evaluated functions at runtime. This permits rapid, easy-to-read definition of product creation pipelines without modifying core application code. This reduces risks associated with code changes, and can even permit end users with little programming experience to modify individual pipeline executions without ongoing side effects. This idiom shares some similarities with both Adobe Photoshop actions and `scikit-learn Pipeline` objects.

Here is a simple instruction for constructing decorrelation stretch (DCS) images from an arbitrary instrument's "R6", "R3", and "R1" bandpasses:

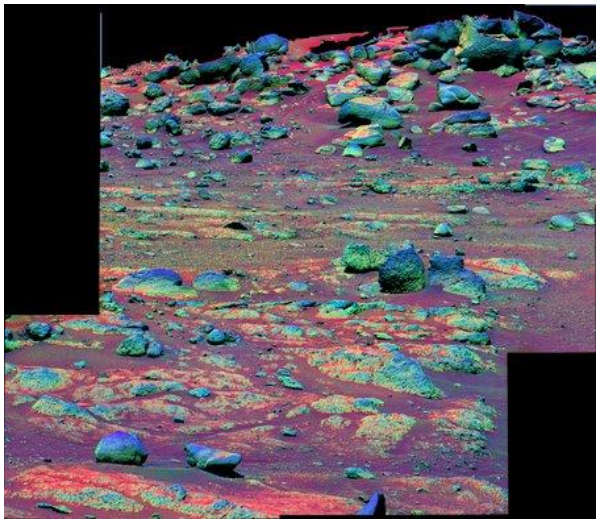
```
DCS = {
    "look": "dcs",
    "bands": ("R6", "R3", "R1"),
    "params": {"contrast_stretch": 1}
}
```

An application could either explicitly call a 'compiled' version of this instruction (possibly after modifying it inline) or pass it to the `make_look_set` method of a `marslab.bandset.BandSet` object, which would implicitly add the `BandSet`'s data and metadata to the instruction's execution context before calling it.

Sky and Shadow Masking: Many of the data representations favored by multispectral analysts are extremely sensitive to input conditions. DCS, for instance, is effective at highlighting spectral diversity, but its intensification of hue differences comes at the price of suppressing value differences. This means that it amplifies noise, suppresses form and line, and can produce wildly different hues for similar scenes if, e.g., one has a small patch of bright sky and the other does not. These effects are longstanding 'pain points' for many analysts.

We initially believed these problems to be poorly tractable given reasonable levels of effort, but have

discovered effective and relatively simple techniques for masking visually and statistically confusing scene elements. (See `marslab.imgops.masking.`) In Mars rover scenes, shadows and the sky are the most common offenders (rover hardware is also very bad but less common). We have found that an *absurdly* simple threshold cut is effective for producing qualitatively satisfying shadow masks in the majority of scenes imaged by MCAM and ZCAM. We have also found that a slightly more complex–but still much simpler than we initially imagined–masking technique is effective for excluding the Martian sky from most scenes. This technique relies on applying a series of morphological operators in series with a Canny operator to generate an image 'trace array', supplementing that trace with a threshold mask, selecting sky-like regions from a labeled version of the composite mask, then 'trimming' the mask before applying it to the initial image(s).



Masked, mosaicked VIS-NIR ZCAM DCS from sol 139.

Masking subsystem. We have implemented a subsystem in `Look` to add these (and other) masking operations to image pipelines using `Composition's` ability to manipulate execution order and data flow with 'sends'. Here is a simple shadow mask definition:

```
SHADOW_MASK = {
    "function": threshold_mask,
    "params": {
        "percentiles": (8, 100),
        "operator": "mean"
    },
    "pass": True,
    "send": True,
}
```

One or more mask definitions of this type can be concatenated into a list and (re)used in other pipelines. The "pass" parameter instructs the `Look` compiler to create a `Composition` that "passes" the mask into subsequent portions of the pipeline (e.g., for masks that remove photometrically invalid portions of an image); the "send" parameter instructs the `Look` compiler to create a `Composition` that applies the mask (possibly with some transparency) to the final rendered image.

Mosaicking: Mosaicked versions of images can provide crucial context for analysts. Unfortunately, creating mosaicked versions of multispectral rover images has historically been quite labor-intensive, so multispectral mosaics have rarely been available on a tactical timeline.

We have found that the popular open-source panorama-stitching package Hugin [5] is fairly successful at producing qualitatively-acceptable mosaics of multispectral rover imagery. Hugin is a complex "black box" processing chain designed for compositing terrestrial images taken with consumer cameras for aesthetic or commercial purposes, so there are some limitations associated with this technique:

Assumptions about image content. Compared to terrestrial images, Mars images have relatively poor chroma information and few line features, so out of the box, it *often* fails. We apply a simple set of photometric and morphological enhancement filters to input images (that we then discard after extracting its geometric mapping) that *usually* allows Hugin's processing to work. Nevertheless, sometimes it *simply does not work*, and it is extremely difficult to determine why. However, working *most* of the time is still a significant improvement on the status quo.

Unsuitability for quantitative analysis. Even well-understood mosaicking processes can introduce radiometric and photometric uncertainties and are discouraged from use as quantitative products; while this methodology is advantageous due to the speed at which it can produce suitable mosaicked images, it too is unsuitable for quantitative analysis, partly due to its black-box nature. We mitigate the opacity of the technology by producing geometric maps indicating the contribution of individual image frames to the mosaic and ensuring that all source images are consistently and immediately available to analysts.

Acknowledgments: Funding provided by the Mastcam-Z instrument investigation and the MSL Participating Scientist Program.

References:

[1] M. Rice et al. (2022) JGR: Planets.

[2] A. Vaughan et al. (2023) JGR: Planets.

[3] C. C. Million et al. (2022) *LPSC VIII,* Abstract #2533.

[4] marslab-reference 'index' repository.

[5] P. d'Angelo et al. Multiple years. No preferred citation.