dfreproject: A Python package for astronomical reprojection

Carter Lee Rhea [1,2*], **Pieter van Dokkum**[1,3*], **Steven R. Janssens** [1], **Imad Pasha**[1,3], **Roberto Abraham**[1,4,5], **William P. Bowman** [1,3], **Deborah Lokhorst**[1,6], **and Seery Chen**[1,4,5]

1 Dragonfly Focused Research Organization, 150 Washington Avenue, Santa Fe, 87501, NM, USA **2** Centre de Recherche en Astrophysique du Québec (CRAQ), Québec, QC G1V 0A6, Canada **3** Astronomy Department, Yale University, 219 Prospect St, New Haven, CT 06511, USA **4** David A. Dunlap Department of Astronomy & Astrophysics, University of Toronto, 50 St. George Street, Toronto, ON M5S 3H4, Canada **5** Dunlap Institute for Astronomy & Astrophysics, University of Toronto, 50 St. George Street, Toronto, ON M5S 3H4, Canada **6** NRC Herzberg Astronomy & Astrophysics Research Centre, 5071 West Saanich Road, Victoria, BC V9E 2E7, Canada ***** These authors contributed equally.

Summary

Deep astronomical images are often constructed by digitally stacking many individual sub-exposures. Each sub-exposure is expected to show small differences in the positions of stars and other objects in the field, due to the movement of the celestial bodies, changes/imperfections in the opto-mechanical imaging train, and other factors. To maximize image quality, one must ensure that each sub-exposure is aligned to a common frame of reference prior to stacking. This is done by reprojecting each exposure onto a common target grid defined using a World Coordinate System (WCS) that is defined by mapping the known angular positions of reference objects to their observed spatial positions on each image. The transformations needed to reproject images involve complicated trigonometric expressions which can be slow to compute, so reprojection can be a major bottleneck in image processing pipelines.

To make astronomical reprojections faster to implement in pipelines, we have written `dfreproject`, a Python package of GPU-optimized functions for this purpose. The package's functions break down coordinate transformations using gnomonic projections to define pixel-by-pixel shifts from the source to the target plane. The package also provides tools for interpolating a source image onto a target plane with a single function call. This module follows the FITS and SIP formats laid out by Greisen & Calabretta (2002), Calabretta & Greisen (2002), and Shupe et al. (2005). Compared to common alternatives, `dfreproject`'s routines result in speedups of up to 20x when run on a GPU and 10x when run on a CPU.

Statement of need

Several packages already exist for calculating and applying the reprojection of a source image onto a target plane, such as `reproject` (Robitaille et al., 2020) or `astroalign` (Beroiz et al., 2020). These packages excel at easy-to-use, general-purpose astronomical image reprojection, but they function solely on CPUs and can be computational bottlenecks in some data reduction pipelines. The `dfreproject` package harnesses GPUs (using `PyTorch` as its computational backbone, Paszke et al., 2019) to improve computational efficiency. The package has minimal reliance on pre-existing WCS packages such as those found in `astropy` or `WCSLIB` (Astropy Collaboration et al., 2013, 2018, 2022), with such packages being used only for convenience in non-computationally expensive steps (such as using `astropy.wcs` to read the header information

from the input fits files).

`dfreproject`'s primary purpose is to reproject observations taken by a new version of the Dragonfly Telephoto Array that is currently under construction in Chile. The volume of data obtained with this telescope will be large, with >1000 exposures obtained simultaneously, and it is paramount that the processing pipeline incorporates fast and accurate reprojection methods. As shown below, by leveraging `PyTorch` for vectorization and parallelization via the GPU, `dfreproject` achieves a considerable speedup (up to nearly 20x) over standard methods.

`dfreproject` can be used as a direct replacement for `reproject.reproject_interp` by simply importing `dfreproject` instead of `reproject` such as:

```python
from dfreproject import calculate_reprojection
reprojected = calculate_reprojection(
    source_hdus=source_hdu,
    target_wcs=target_wcs,
    shape_out=target_hdu.data.shape,
    order='bilinear',
    device='cpu',
    n_threads=2
)
```

The `target_wcs` argument can be passed as a header similar to `reproject`. Additionally, if `shape_out` is not provided, the shape will be the same as the input.

Methods

We must perform three intermediate calculations to reproject an image onto a new coordinate plane. To do this, we use the target and source WCS.

Before defining the steps, there are a few terms to define:

- Standard Imaging Polynomial (SIP). This convention allows us to represent non-linear geometric distortions as a simple polynomial. The order and coefficients of this polynomial are stored in the header. The SIP is broken down into four individual polynomials, SIP_A, SIP_B, SIP_INV_A, and SIP_INV_B where SIP_A defines the polynomial applied to the x-coordinates, SIP_B defines the polynomial applied to the y-coordinates, and SIP_INV_A and SIP_INV_B define the inverse operations. For an in-depth discussion on SIP, see Shupe et al. ([2005](#)).

- Coordinate Description (CD) Matrix. This is a 2x2 matrix that encodes the image's rotation, skew, and scaling. The values are conveniently stored in the header. The CD matrix may also be constructed from the PC Projection Coordinate matrix multiplied by the CDELT values.

The steps are as follows:

1. For each pixel, calculate the corresponding celestial coordinate using the target WCS
   a. Apply shift
   b. Apply SIP distortion
   c. Apply CD matrix
   d. Apply transformation to celestial coordinates using the Gnomonic projection
2. Calculate the position in the source grid for each celestial coordinate. This provides the offset for the next step.
   a. Apply the inverse transformation using the Gnomonic projection
   b. Apply inverse CD matrix
   c. Apply inverse SIP distortion
   d. Apply shift
3. Interpolate the source image onto the newly calculated grid

In the final interpolation step, we include local flux conservation by simultaneously projecting an identity tensor called the footprint. The final reprojected frame is normalized by this footprint.

Results

We created two small (50x50) FITS files for this demonstration with a pixel offset of 0.5 pixels and a 0.005 degree offset in the header. In Figure 1 from left to right, we show the original image, the `dfreproject` solution, the `reproject` solution, and the relative error between the two. We define the relative error as $100 \cdot \frac{\text{dfreproject\_solution} - \text{reproject\_solution}}{\text{reproject\_solution}}$. For both solutions, we use a bilinear interpolation scheme. In the noisy regions of the image, the differences in the reprojections is pure noise. There are slight differences in the solutions at the locations of the Gaussians which is attributable to small differences in the normalization. In this demo, we used the GPU by setting `device='gpu'` in the argument of `calculate_reprojection()`.
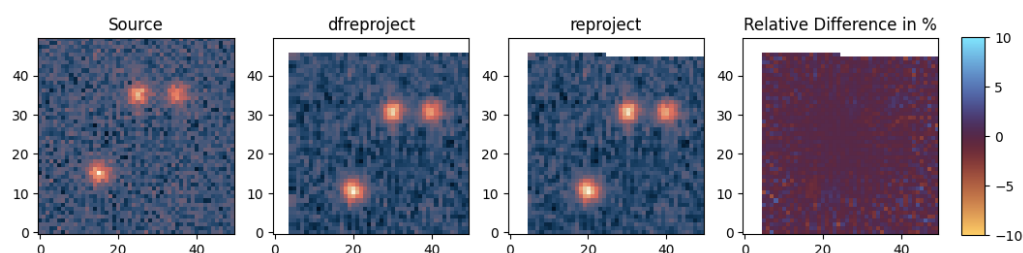

Figure 1: Comparison between `dfreproject` and `reproject`.

To compare the execution times, we created a benchmarking script (which can be found in the demos/benchmarking directory under `benchmark-script.py`; the figures are constructed with `benchmark-plotter.py`). This test is run between `dfreproject` and `reproject`. We benchmark the three interpolation schemes with and without SIP distortion for images sized 256x256 to 4000x6000 (this approximately matches the size of Dragonfly images). Figure 2 shows the results of this benchmarking when `dfreproject` is run using a GPU (NVIDIA GeForce RTX 4060).
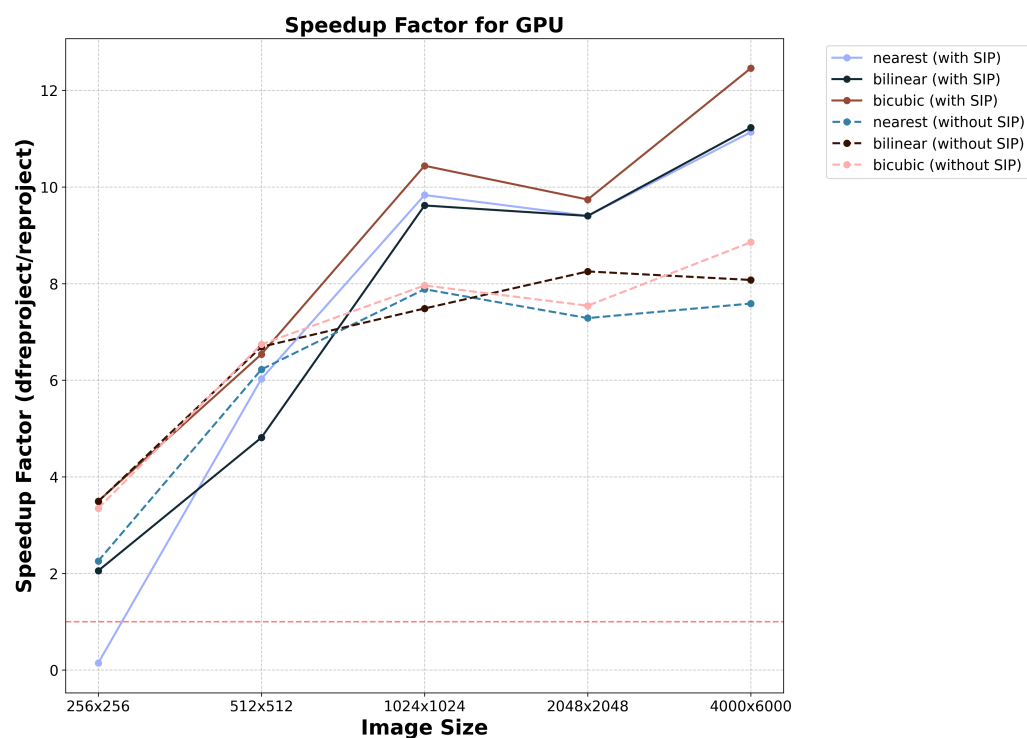
Figure 2: Benchmarking results for `dfreproject` on a GPU.

As evidenced by this figure, `dfreproject` has a significant speed advantage over `reproject` for larger images regardless of the type of interpolation scheme. The speedup is most pronounced in the case where SIP distortions are included.

In Figure 3, we display the same results except we used a CPU (Intel® Core™ i9-14900HX). For this experiment, we used a single thread.
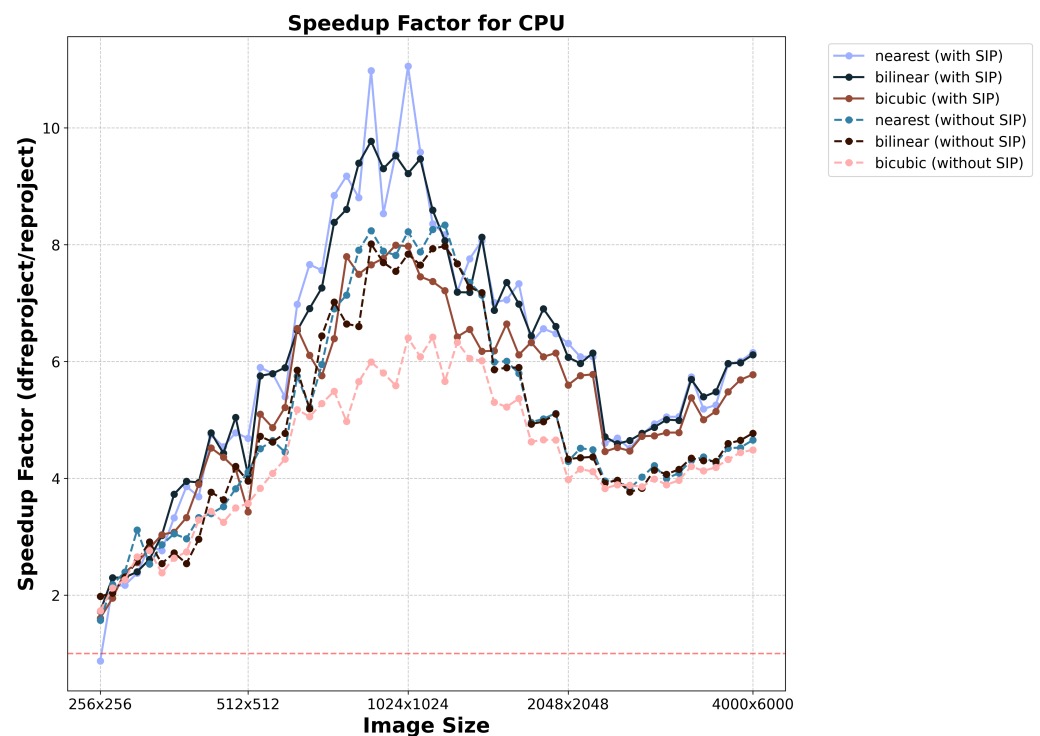
Figure 3: Benchmarking results for `dfreproject` on a CPU.

Although the speedup on the CPU is not as impressive as on the GPU, it is still considerable. All code can be found in the demo directory.

Acknowledgements

References

Astropy Collaboration, Price-Whelan, A. M., Lim, P. L., Earl, N., Starkman, N., Bradley, L., Shupe, D. L., Patil, A. A., Corrales, L., Brasseur, C. E., Nöthe, M., Donath, A., Tollerud, E., Morris, B. M., Ginsburg, A., Vaher, E., Weaver, B. A., Tocknell, J., Jamieson, W., … Astropy Project Contributors. (2022). The Astropy Project: Sustaining and growing a community-oriented open-source project and the latest major release (v5.0) of the core package. *The Astrophysical Journal*, *935*, 167. https://doi.org/10.3847/1538-4357/ac7c74

Astropy Collaboration, Price-Whelan, A. M., Sipőcz, B. M., Günther, H. M., Lim, P. L., Crawford, S. M., Conseil, S., Shupe, D. L., Craig, M. W., Dencheva, N., Ginsburg, A., VanderPlas, J. T., Bradley, L. D., Pérez-Suárez, D., Val-Borro, M. de, Aldcroft, T. L., Cruz, K. L., Robitaille, T. P., Tollerud, E. J., … Astropy Contributors. (2018). The Astropy Project: Building an open-science project and status of the v2.0 core package. *The Astronomical Journal*, *156*, 123. https://doi.org/10.3847/1538-3881/aabc4f

Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., Greenfield, P., Droettboom, M., Bray, E., Aldcroft, T., Davis, M., Ginsburg, A., Price-Whelan, A. M., Kerzendorf, W. E., Conley,

A., Crighton, N., Barbary, K., Muna, D., Ferguson, H., Grollier, F., Parikh, M. M., Nair, P. H., … Streicher, O. (2013). Astropy: A community Python package for astronomy. *Astronomy and Astrophysics*, *558*, A33. https://doi.org/10.1051/0004-6361/201322068

Beroiz, M., Cabral, J. B., & Sanchez, B. (2020). Astroalign: A Python module for astronomical image registration. *Astronomy and Computing*, *32*, 100384. https://doi.org/10.1016/j.ascom.2020.100384

Calabretta, M. R., & Greisen, E. W. (2002). Representations of celestial coordinates in FITS. *Astronomy and Astrophysics*, *395*(3), 1077–1122. https://doi.org/10.1051/0004-6361:20021327

Crameri, F. (2023). *Scientific colour maps*. Zenodo. https://doi.org/10.5281/zenodo.8409685

Greisen, E. W., & Calabretta, M. R. (2002). Representations of world coordinates in FITS. *Astronomy and Astrophysics*, *395*, 1061–1075. https://doi.org/10.1051/0004-6361:20021326

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). *PyTorch: An imperative style, high-performance deep learning library*. https://doi.org/10.48550/arXiv.1912.01703

Robitaille, T., Deil, C., & Ginsburg, A. (2020). reproject: Python-based astronomical image reprojection. *Astrophysics Source Code Library*, ascl:2011.023. https://ui.adsabs.harvard.edu/abs/2020ascl.soft11023R

Shupe, D. L., Moshir, M., Li, J., Makovoz, D., Narron, R., & Hook, R. N. (2005). The SIP convention for representing distortion in FITS image headers. In P. Shopbell, M. Britton, & R. Ebert (Eds.), *Astronomical Data Analysis Software and Systems XIV* (Vol. 347, p. 491). https://ui.adsabs.harvard.edu/abs/2005ASPC..347..491S