Image Processing in Python With Montage

John Good,[1] and G. Bruce Berriman[2]

[1]*Caltech/IPAC-NExScI, Pasadena, CA 91125, USA;* jcg@ipac.caltech.edu

[2]*Caltech/IPAC-NExScI, Pasadena, CA 91125, USA*

Abstract.

The Montage image mosaic engine (http://montage.ipac.caltech.edu; https://github.com/Caltech-IPAC/Montage) has found wide applicability in astronomy research, integration into processing environments, and is an examplar application for the development of advanced cyber-infrastructure. It is written in C to provide performance and portability. Linking C/C++ libraries to the Python kernel at run time as binary extensions allows them to run under Python at compiled speeds and enables users to take advantage of all the functionality in Python. We have built Python binary extensions of the 59 ANSI-C modules that make up version 5 of the Montage toolkit. This has involved a turning the code into a C library, with driver code fully separated to reproduce the calling sequence of the command-line tools; and then adding Python and C linkage code with the Cython library, which acts as a bridge between general C libraries and the Python interface.

We will demonstrate how to use these Python binary extensions to perform image processing, including reprojecting and resampling images, rectifying background emission to a common level, creation of image mosaics that preserve the calibration and astrometric fidelity of the input images, creating visualizations with an adaptive stretch algorithm, processing HEALPix images, and analyzing and managing image metadata.

The material presented here will be made freely available as a set of Jupyter notebooks posted on the Montage GitHub page.

1. Introduction

- Montage - image mosaic engine. Creates mosiacs from input set of FITS images Written in ANSI -C. Portable. Components perform one task in the creation of a mosaic (list them). Plus utilities for managing and organizing files, managing FITS attributes, and analyzing image metadata. List them

- Wide appplicability. Used in NEO detection, Instrument Performance, Observation planning for JWST, Citizen Science, Machine Learning.

- . Created Python binary extensions of 59 modules in v5 of Montage. Gives users the power of Python at compiled speeds. This has involved a turning the code into a C library, with driver code fully separated to reproduce the calling sequence of the command-line tools; and then adding Python and C linkage code with the Cython library, which acts as a bridge between general C libraries and the Python interface.

The Python extensions have been released as v6 on Nov 12 2018. tested on Python 3.6, Mac OS X.

Python binary extensions of existing Montage modules; no new functionality has been introduced. Click here to see a list (link to Jupyter notebook) of all supported modules. The Python extensions have been created by transforming the C code (https://github.com/Caltech-IPAC/Montage) into a library, with driver code fully separated to reproduce the calling sequence of the command-line tools; and then adding Python and C linkage code with the Cython library, which acts as a bridge between general C libraries and the Python interface. These binary extensions offer image processing at compiled speeds in the Python environment.

2. How To Install and Use It

There are two ways to install MontagePy, all of which include all supporting packages: there are no external dependencies. From PyPI, use the command "pip install MontagePy." Or download the .whl file (add link) and install with the command "pip install file.whl."

We have delivered a set of Jupyter notebooks that give examples of how to use each component in Python, and compares usage in Python with that in C. The Jupyter notebooks are available for download at https://github.com/Caltech−IPAC/MontageNotebooks and they can be viewed without downloading at http://montage.ipac.caltech.edu/MontageNotebooks.

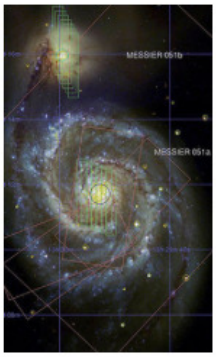
3. Building A Mosaic With Python

4. Visualizing Images in Python with Montage

Acknowledgments. Montage is funded by the National Science Foundation under Grant Numbers ACI-1440620 and ACI-1642453., and was previously funded by the National Aeronautics and Space Administration's Earth Science Technology Office, Computation Technologies Project, under Cooperative Agreement Number NCC5-626 between NASA and the California Institute of Technology.

5. References

Building a Mosaic with Montage



Montage is a general toolkit for reprojecting and mosaicking astronomical images and generally you have to marshall the specific data you want to use carefully. But there are a few large-scale uniform surveys that cover a large enough portion of the sky to allow a simple location-based approach.

In this notebook we will choose a region of the sky and dataset to mosaic, retrieve the archive data, reproject and background-correct the images, and finally build an output mosaic. You are free to modify any of the mosaic parameters but beware that as you go larger all of the steps will take longer (possibly **much** longer). If you do this for three different wavelenths, you can put them together in a full-color composite using our Sky Visualization notebook, which produced the image on the right.

As with many notebooks, this was derived from a longer script by breaking the processing up into sequential steps. These steps (cells) have to be run one in sequence. Wait for each cell to finish (watch for the step number in the brackets on the left to stop showing an asterisk) before starting the execution of next cell or run them all as a set.

If you want to just see the code without all the explanation, check out this example.

Setup

The Montage Python package is a mixture of pure Python and Python binary extension code. It can be downloaded using `pip install MontagePy`

No other installations are necessary.

In [3]:
```
# Startup.  The Montage modules are pretty much self-contained
# but this script needs a few extra utilities.

import os
import sys
import shutil

from MontagePy.main    import *
from MontagePy.archive import *

from IPython.display import Image


# These are the parameters defining the mosaic we want to make.

location   = "M 17"
size       = 1.0
dataset    = "2MASS J"
workdir    = "Messier017"
```

So not much to see so far. We've defined a location on the sky (which can be either an object name (*e.g.* `"Messier 017"`) or coordinates. The coordinate parser is pretty flexible; `"3h 29m 53s +47d 11m 43s"` (defaults to the Equatorial J2000 system), `"201.94301 47.45294 Equ B1950"` and `"104.85154 68.56078 Galactic"` all work. We've also defined a size. In this case we are going to use this below to construct a simple North-up gnomonic projection square box on the sky; you are free to define any header you like as Montage supports all standard astronomical projections and coordinate systems.

Working Environment

Before we get to actually building the mosaic, we need to set up our working environment. Given the volume of data possible, the Montage processing is file based and we need to set up some subdirectories to hold bits of it. This will all be under an instance-specific directory specified above ("workdir"). It is best not to use directory names with embedded spaces.

In [4]:
```
# We create and move into subdirectories in this notebook
# but we want to come back to the original startup directory
# whenever we restart the processing.
```

Figure 1. Section of Jupyter Notebook for Building a Mosaic of M17

In [16]:
```
try:
    os.mkdirs('work/SDSS')
except:
    pass

rtn = mViewer(imgjson, 'work/SDSS/SDSS.png', mode=1)

print(rtn)
```


```
{'status': '0', 'type': b'color', 'nx': 1200, 'ny': 1200, 'grayminval': 0.0, 'grayminpercent': 0.0, 'gr
ayminsigma': 0.0, 'graymaxval': 0.0, 'graymaxpercent': 0.0, 'graymaxsigma': 0.0, 'blueminval': 1660.948
886289596, 'blueminpercent': 45.51304816901208, 'blueminsigma': -0.10000000000002897, 'bluemaxval': 188
33.51097929482, 'bluemaxpercent': 100.0, 'bluemaxsigma': 7293.27099365378, 'greenminval': 612.19766982
25872, 'greenminpercent': 43.049856485589686, 'greenminsigma': -0.1000000000000262, 'greenmaxval': 2627
6.085778495264, 'greenmaxpercent': 100.0, 'greenmaxsigma': 14790.776567073211, 'redminval': 896.9336930
051037, 'redminpercent': 42.62543763181061, 'redminsigma': -0.10000000000000908, 'redmaxval': 20567.639
87893109, 'redmaxpercent': 100.0, 'redmaxsigma': 6285.599891788824, 'graydatamin': 0.0, 'graydatamax':
0.0, 'bdatamin': 1650.1333317872777, 'bdatamax': 18833.51097929482, 'gdatamin': 608.5073366952034, 'gda
tamax': 26276.085778495264, 'rdatamin': 890.1481250441316, 'rdatamax': 20567.63987893109, 'flipX': 0, '
flipY': 1, 'colortable': 0, 'bunit': b''}
```


The return contains details of the individual image stretching, image size, and so on.

Here is the output image:

In [17]:
```
from IPython.display import Image

Image(filename='work/SDSS/SDSS.png')
```

Out[17]:

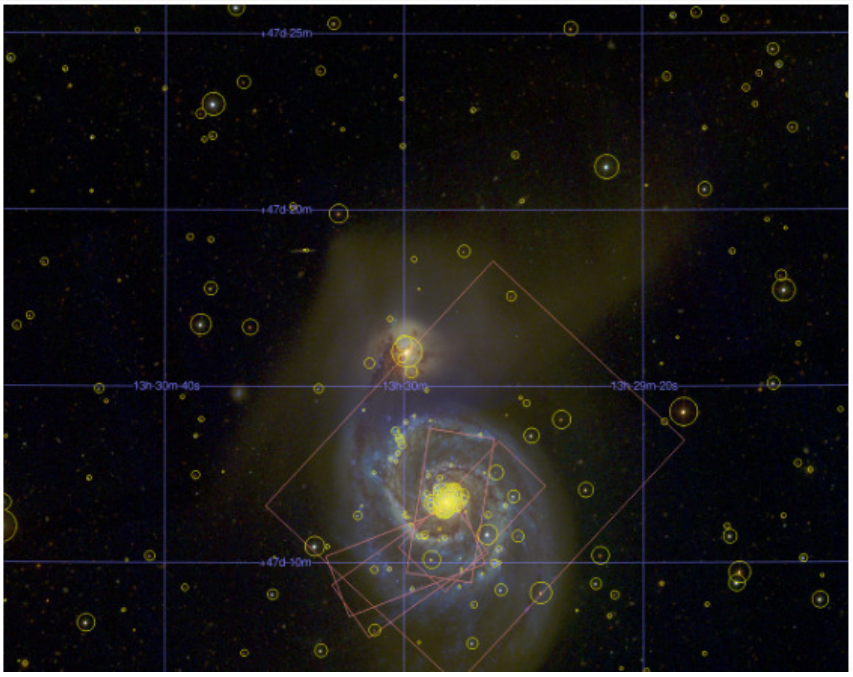


Figure 2. Three color Slaon Image of M51 Created With mViewer