**“Image Pop”**

**Baby Steps Towards A New Implementation of**

**Practical Parallel File Processing**

**For**

**Astronomical Data Acquisition, Storage and Manipulation**

Written in Python From the ground up

Sol Courtney, Lowell Schudel

December 23, 2015

*Department of Astronomy and Astrophysics, Columbia University, New York City, New York*

**Contents:**

**1. Introduction**

**1.2 Motivations**

**1.3 Benefits Provided To The User**

**2. General Descriptions and Content Overview**

**2.2 Software**

**2.3 Hardware**

**2.4 Data**

**2.5 Demonstration**

**3. Demonstrating Operability**

**3.2 Choosing a Directory, Downloading & Set Up.**

**3.3 Initializing Your Directory**

**3.4 The Configuration File**

**3.5 Running ImagePop.py**

**3.6 Supplying Mock-Data-Set**

**3.7 Understanding The Output**

**4. Appendix**

**1**

**- Introduction -**

This paper has been written for the purpose of fulfilling the requirement of the fall 2015 Observational Astronomy ASTROW3646x final project. This paper will describe in detail the inner workings of a newly developed image-processing program named ImagePop.py and its companion library ImagePopLib. Included will be a semi-specific description of the computer and network hardware which facilitate the program for this particular demonstration, a detailed description of the data set used for this demonstration and a step-by-step outline of the program’s intended workflow as performed herein. This paper covers the first of three steps towards realizing a far more useful and versatile program, capable of executing the basic functions presented here but in a fully parallel framework. The final version of this application will not only be computationally superior but will also offer the user more useful features and will be far more fault tolerant.

The program ImagePop.py is currently developed and supported by the authors of this paper, Lowell Everest Schudel and Sol Weatherford Courtney, both students in the Department of Astronomy and Astrophysics at Columbia University in New York City New York. It is our hope for this program to be openly developed and contributed to by the larger Astronomical society and serve as a simple but useful tool for anyone whom finds him or herself at work under a telescope.

- 1.2 Motivations -

This project is driven by the notion that our data today already hold answers to questions we yet not know to ask. We believe proper exploitation of data depends largely on a cultivation-like practice of acquisition and preservation. During our time at the MDM Observatory we realized the benefit that low level networked parallel image manipulation workflow applications would have on the larger and more general image capturing process. For instance, a large number of low-cost single-board-computers could be tasked with an appreciable workload without the overhead and limiting exclusiveness of a typical cluster computer. Our project then is to create the single-machine version of such a distributed application.

We thought this improvement would provide a faster route to properly sorted and catalogued personal data sets with little to no extra expense to the user. It was then, and is still now, our belief that simple software based automated data sorting and manipulation will realize a benefit to virtually any that elect its utilization. We hope that by providing this application we can make a small but positive contribution to the much larger ongoing process of scientific exploration.

- 1.3 Benefits provided to the user -

It is our belief that the intended user will benefit in several ways from using this application. Firstly the user no longer needs to manually perform such repetitive tasks such as RGB calibration, .png thumbnail creation, and many other common or specialized processes witch are typically performed on the majority of “fits” data. The user only needs to know the operations ahead of time to profit from ImagePop. We understand that most Astronomers capturing images theses days already have a basic set of initial operations they routinely perform on all images. By selecting this “base-line” set of operations understood as standard to the data set before collection begins, initializing ImPop and nothing more, the user arrives at the same place as he/she would have otherwise but will instead this way realize the same progress having taken considerably fewer steps and interacted with the computer substantially less.

Secondly the user benefits from the effortless record-keeping function of ImagePop. All functions and operations regardless of complexity, will be chronologically recorded and can be, at any time, referenced by the user at will. This is a principal concern for consistency over long periods of time. The logging and recording of repetitive tasks associated with long-term data collection is an easy place for important information to be lost. ImagePop covers that potential loss for its users.

Thirdly we suppose our users will gain from ImagePop’s versatility. The software is pure Python, it has uncomplicated features, it’s ready to accept almost any custom operation the user may wish to add into its library without fuss. The user can easily run simultaneous instances of the program on a single data set. Above all it is being developed openly and free to use for all.

For these reasons stated above and for several others not yet mentioned, we feel confident that ImagePops’ performable efficacy, its value to the user and its quality as an academic exercise make it a worthwhile endeavor and an ideal undertaking for students like ourselves. While ImagePop is not complex or deeply innovative, both its plausible utility and the reasonable amount of effort required to develop it, are the reasons why, for this assignment, we decided to make it into something real for everyone to share.

**2**

* **General Descriptions & Content Overview -**

All of the required items needed to preform this demonstration are available from the GitHub Repository [github.com/Schudel888/ImagePop](http://github.com/Schudel888/ImagePop) and should be downloaded together if a reproduction of this demonstration is intended. The system requirements for this demonstration include an operating workstation equipped with an implementation of Python version 2.7, all Python modules listed in the appendix, a program for viewing “fits” files if the user wishes to inspect the output, a working internet connection and a basic understanding of the terminal.

All questions and comments should be sent to either [swc2124@columbia.edu](mailto:swc2124@columbia.edu) or [les2185@columbia.edu](mailto:les2185@columbia.edu) to be addressed.

- 2.2 Software -

ImagePop.py is the only program the user needs to run. By selecting arguments, the user can access every possible function provided to ImagePop.py by the user. The main two arguments are ‘init’ and ‘run’ either of which will always be present in the users’ command. Before ImagePop can accept the ‘run’ argument, the user must select a directory to be initialized by the ‘init’ argument. This directory is supposed to be the same directory the user will later save image data to and is referred to as the “target-directory”. ImagePop.py can only run if an initialized directory exits and the path to this directory has been supplied to ImagePop.py.

**Note:** *The target directory contains all of the user input and is the directory that must be initialized. The parent directory is the directory where ImagePop is run from. The parent directory does not need to be initialized.*

After a target-directory has been initialized the target-directory will then contain several unique files to be used by both the programs’ functions and the user for various aspects of operation. These newly created files within the newly initialized target-directory are all itemized and defined in the appendix and will be discussed later, as they become germane for the purpose of our demonstration.

The operations, which ImagePop.py can perform on the data, are intended, but not required, to be listed within the same directory as ImagePop.py and ImagePopLib.py. By following only a few simple conformal requirements, the user can add operations to the library without limit. The operations which ImagePop.py will make use of for this demonstration are all listed in the appendix and can be referenced there. Each operation will be explained as they are called in the demonstration. There is a template for user-defined-operations inside of the repository (png.py, starfind.py) and at the top of the ImagePopLib.py file as well.

- 2.3 Hardware -

The goal of this project is to eventually develop a fully parallel application capable of utilizing multiple machines at once to achieve the same task as we are achieving here. This application would be equipped to access and assess the resources available to it and deploy them appropriately. For this demonstration, we present a single machine version. This version is fundamentally the same as the parallel application except for the programs’ ability to operate simultaneously without error. Because this is an Observational Astronomy project and not a computer science course, we elected to invest our effort into the programs’ more appropriate areas. In the future we intend to develop the parallel version of our application, which we expect to be far more powerful.

- 2.4 Data -

It is our belief that ImagePop can potentially accept many data types- not just “fits” files. For this demonstration we will be using a collection of preselected “fits” files sourced from our time working with both the Hiltner 2.4 meter and the McGraw 1.3-meter scope at the MDM Observatory in Arizona. For this demonstration we have created a simple bash script that serves as a proxy for the actual “data-flow” that would be entering the target directory over the entire observation period. This bash script is available from the repository and is included in the download.

We will not use spectrographic images nor will we use any image in which stars cannot be rendered as point sources, because the demonstrations makes use of functions which are tailored to such images. This collection of data we are using is then very simple and is of course, not nearly the extent of ImagePops’ potential. We have selected image files, which fit the needs of the operations we currently have to present, as the number of operations increase and as the library develops, the amount of data needed to test the programs’ operable aspects will increase and we hope it does. For now, this is the data set we are using. It includes an exo-planetary solar transit, a mosaic of the Orion Nebula, several disk galaxies, many star fields, the Carina Nebula and numerous similar images.

The collection of “fits” files used here are images captured with a single sensor that was moved and used on two separate telescopes. The Hiltner 2.4 meter scope and the McGraw 1.3 meter scope both located on Kitt Peak National Observatory in Arizona, USA were used to collect all images files in the mock-data-set. The device used on both telescopes is called OSU MDM4K and it was fitted along with another device called OSMOS (Ohio State Multi-Object Spectrograph). MDM4K website is here: <http://www.astronomy.ohio-state.edu/MDM/MDM4K/>. OSMOS users manual website is here: <http://www.astronomy.ohio-state.edu/~martini/osmos/>. The instruments are capable of capturing far more data rich images than are being used here and in no way is this demonstration or paper meant to contend with, or even speak to, the far outstanding capabilities of either the OSU MDM4K sensor or the OSMOS device.

We simply supply our initialized target-directory with the mock-data and ImagePop goes to work just the same. This way we can intentionally tweak the incoming data for the purpose of testing ImagePops’ basic resilience to fault. All files and data types used for this demonstration are, of course, listed in the appendix with all the other items of interest.

- 2.5 Demonstration -

The demonstration video will cover everything from the program download to reviewing the format of outputted data. The video is accessible on YouTube at <https://www.youtube.com/watch?v=k1fDdByZWY8> and can be acquired by contacting either of this papers’ authors. While the demonstration is intended for grading, the content covered in both the video and the paper and are intended to serve both as installation guides and basic user manuals for future users.

**3**

**- Demonstrating Operability -**

All steps required for reproduction of our results will be covered in the following sections. Each section is arranged as to partition the total process into sensible subsections. As mentioned earlier, you will need an active Internet connection with the ability to operate git or a hard copy of the required material if you are to follow the steps presented bellow. Again, all necessary material can be gained by either contacting one of the authors or by cloning the provided git repository. Your station needs to have a Python 2.7 distribution and all Python Modules listed in the appendix installed and working. As we have mentioned earlier, what follows is the installation and operation guide to a single computer implementation of, what we are implying is, a program more fully realized as a parallel application, although at this point of development, exists only as a single machine application. In other words, this is the complete first sensible step towards a clearly understood much larger goal; a fully parallel version of this program.

- 3.2 Choosing a Directory, Downloading and Set Up -

The first step, after all system requirements are met, is to select a directory to serve as the proxy data-save directory. This directory serves as a representation of the would-be directory wherein the actual data would be saved. Normally this directory would be saved to over the course of days or weeks but in this case it will be a much shorter period. The user can make this directory anywhere in their file system but for ease of use and general tidiness we suggest first making a top-level directory named ~/ImagePop. Within this directory we can make and do everything we need.

**[STEP 1]:** From the terminal run the following:

cd

git clone <https://github.com/Schudel888/ImagePop.git>

cd ImagePop

mkdir ImagePopTarget

ls –alF

This may take several minutes depending on your connection strength. If the download was successful you should see a new directory named ImagePop. Once the download is complete and all files are copied to the ImagePop directory, we need to add an alias to the local .bashrc (assuming a Unix shell). This step is not required for usage but is recommended because it makes things simpler as we go.

To edit the .bashrc file, run the following commands where nano could be any suitable text editor:

cd

sudo nano .profile

At the bottom of the file add the lines:

alias impop=’python ~/ImagePop/ImagePop.py’

<ctr-x> to exit and <y> to save

After exiting and saving, restart the terminal session and reenter the ImagePop directory. For users unable to make this alias, simply use the command in quotes in place of impop.

- 3.3 Initializing your Directory -

ImagePop.py knows two types of directories, the “parent-directory” and the “target-directory.” The directory where data will be saved is referred to as the target directory. ImagePop.py can only run if the target directory is an initialized directory. The “parent-directory” is the directory where impop is being run from, of which the target directory must be a child. We will be running impop from the newly created ImagePop directory. Initializing a directory allows impop to operate within that directory; it provides the awareness of the application insofar as what operations have been called and on what files. The parent-directory allows the user to access the user configuration file, the Readme, and user defined operations. If the user attempts to ‘run’ impop with an un-initialized directory as a target-directory, impop will automatically revert to the ‘init’ function and ask if the user intended to initialize the directory. The next step then is to initialize the ImagePopTarget directory and confirm our command was successful.

**[STEP 2]:** From the ~/ImagePop directory, run the following:

impop init ImagePopTarget

Regardless of weather this command returns successfully, one file will appear in the parent-directory:

1. **ImagePopReadme.txt**

This is not essential to the functions of the demonstration, but provides helpful information to the user.

Once the directory has been successfully initialized via <impop init>, one new file will exist in the target-directory:

1. **ImagePopInit.txt**

A high-level configuration file the user should not modify.

Now that the directory has been initialized we are ready to execute our test run on the test data.

- 3.4 The Configuration File -

When it is not already present in the parent-directory, the default ImagePopConfig.py file will be generated by the user’s first call to <impop –run>. Before any subsequent <run>, the user can modify this configuration file, which will then be obeyed by any call to impop made in this parent-directory. This configuration file is meant to be human readable Python that even a novice user could navigate, but the limitations listed in comments must be obeyed. While not all variables will be discussed in this section, modifying WAIT\_INTERVAL and LEASH\_LENGTH will be required for this demonstration. We need to set the line starting with “WAIT\_INTERVAL” equal to one (1). This is the length of time that impop sleeps between checking the target-directory for newly written files. We need to set the line starting with “LEASH\_LENGTH” equal to six hundred (600). This is the length of time that impop will run autonomously in the target-directory.

**[STEP 3]:** From the terminal run the following commands:

cd ~/ImagePop

impop run ImagePopTarget

sudo nano ImagePopConfig.py

Chang the value of the line starting with “WAIT\_INTERVAL” to 1

Chang the value of the line starting with “LEASH\_LENGTH” to 600

<ctr-x> to exit and <y> to save

The time interval is now set to one second between refreshing during a ten minute runtime, and we can move on to running impop. Under normal operating conditions, a longer WAIT\_INTERVAL and LEASH\_LEAGTH are desired for the purpose of conserving workstations’ computational resources.

- 3.5 Running ImagePop.py -

Although there are no data in the ImagePopTarget directory, we can start impop regardless and later when data is supplied, it will be detected and operated on accordingly. This is in fact the intended method for this application. Before the first image capture occurs, impop is alert and operational, waiting to discover new files saved to its target directory.

The command to run impop will accept arguments. The arguments reference individual operations that the user wishes the application to preform once on all data saved to the target-directory during the time impop is running. The users can pass as many arguments as they like just as long as the ImagePopLib.py file contains the proper linkage to an existing Python script supporting the operation. Here we will be using several supplied operations.

**[STEP 4]:** From the terminal, run the following command:

impop run ImagePopTarget sf

**<impop>** - is the alias we supplied to the .bashrc file for the longer command: **<python ImagePop.py>**

**<run>** - is now where <init> was in our previous command. In the second position either <init> or <run> must be present, always.

**<sf >** - is a supplied operation. It will locate point sources in a “fits” image and return a table of their coordinates.

Now impop is running in this directory, indicated by the presence of <.ImagePopLock>, named by ImagePopConfig.py. Four files and one new directory will now exist in this target-directory:

1. **ImagePopOperations.txt**

The historical list of operations past to impop run, for example <sf>.

1. **ImagePopHistory.csv**

The table of timestamps corresponding to input and output of impop.

1. **ImagePopIndex.txt**

Ordered File List generated into target directory.

1. **.ImagePopLock**

The presence of this file will prevent any simultaneous instance of impop run from being executed with the same targeted directory. This could be changed in future versions but at this stage is necessary.

1. **Sf/**

Output directory created for the operation <sf>. Should contain all output created by the <sf> function.

Now impop is running and looking to the target directory for newly saved image files every second (WAIT\_INTERVAL). For the next ten minutes this terminal will be occupied by impop because it is running autonomously. As soon as we supply ImagePopTarget with the mock data, impop will begin operating on the target-directory and outputting to the <sf> directory. impop will do its best to remain active unless a fatal error occurs for the entire ten minutes (LEASH\_LENGTH). Operation beyond the LEASH\_LENGTH will continue until work ordered earlier completes.

We have now created a running instance of impop and an output directory for the operation <sf>. In the same way as we have created this instance, we will now create a second instance, in a new terminal, with the <sf> output directory as the target-directory.

**[STEP 5]:** From a new terminal, run the following command:

cd ~/ImagePop

impop init ImagePopTarget/sf

impop run ImagePopTarget/sf png

**<png>** - will make a “png” image for each identified file.

Now the second instance of impop is running in this sub-directory, which will contain all the same files as in the previous step, except for a directory called <png> instead of <sf>. This instance will be outputting png files using the <sf> output as input.

3.6 Supplying Mock-Data-Set

Now that ImagePop is up and waiting for new files to be save into the target directory, we can run the supplied bash script named MockDataSender.sh. As long as the test data exist as part of the cloned repository, the script will begin to distribute the “fits” files to the directory named ImagePopTarget so long as they both live in the same directory. If needed, the MockDataSender.sh script can be easily altered by the user.

**[STEP 5]:** In a third terminal, run the following commands:

*\*Optionaly, the user may elect to arrage all terminal windows as to best view the program execution. The authors revomend three vertical terminals arranged sid by side.*

cd ~/ImagePop

MockDataSender

You should now see the output messages of this bash script as it begins a periodic depositing of the mock data set into the ImagePopTarget directory. Now ImagePop will be able to detect this newly saved “fits” data in the top target-directory and will begin performing and saving the outputs of our selected operations.

3.7 Understanding the Output

ImagePop will save the output of each performed operation in a new directory titled with the name of the operation. Within the new directory, each file will have the same name as the original file from which it came, which from now will be referred to as the *designated output file*. For example, if three arguments are passed to ImagePop, then the output will be written in three separate directories all within the target-directory and each of the new output directories will be named according to the operation performed on the files it contains. This directory is the operation’s *output directory* and every operation performed by ImagePop will have one.

Operations may place additional files in their output directory, unique to the operation, which could then be referenced from the designated output file. If an operation is inexpensive and produces no writable output, then the output directory will still be created but will in this case contain no files. It is recommended that if an operation is expensive, or if it requires output of any type, that a placeholder file still be created in the output directory.

ImagePop keeps a segmented record of all activity ever performed within an initialized directory in the form of several text and table files. These files are human readable and can be of great use to the user.

The < **ImagePopConfig.py** > file must be found in the calling directory of impop, or it will be generated on the spot by any call to impop run. Generating the config file in the location of ImagePop.py or ImagePopLib.py would make it apply too broadly to every impop operation on a user’s system, whereas placing it in the target directory would limit reuse between similar projects. Thus the compromise- that it lives in the calling directory, which must be a parent of the target directory but not necessarily the same directory as the source code, was reached. Therefore, the ImagePopConfig.py file should contain information relative to any datasets you intend to process with the same parameters.

The < **.ImagePopLock** > file indicates that a terminal is running impop on this target directory, and that you should not run additional run operations here unless you are ready to override the programs locking mechanisms. Do not modify or tamper with! In rare cases where an error has occurred, the user may wish to delete a stubborn lock that was not removed properly by the program.

The < **ImagePopLog.txt** > file is a rudimentary logging tool that the user can use to track messages from ImagePopLib and the errors caught and handled during the execution of user supplied functions on the input data. Concurrent execution of multiple impop terminals from the same parent directory will share a log file, which may cause asynchronous writes and other bad errors.

The < **ImagePopInit.txt** > file contains machine written data that is intended to allow for future calls to impop run. It can be fleshed out in later versions to facilitate other calls in ImagePop.py or ImagePopLib.py as the writers see fit.

The < OPERATION\_NAME > directory ought to contain at least one output file for each input, with the same name as the original, to indicate the operation completed successfully and does not need to be run again. This directory may contain many more files- which the user will be responsible for organizing, or none at all- which would cause the function to be re-run for every input in every session after every WAIT\_INTERVAL. It is important to design your library function to make use of the existing “one-file-in one-file-out” pattern we have demonstrated. Examples like sf, png, and png\_contour can be found in our youtube demonstration video.

The < **ImagePopIndex.txt** > now contains a list of every input file that has been seen during any run of impop in the target directory, one filename per line. Since it was created by the write\_sealed\_text function of ImagePopLib, it also contains an ImagePopConfig.py specified header that remains separate from any data that follows. It is arbitrarily ordered, but could be sorted chronologically or lexically in a future version of the program. The human user must not modify it, despite it being human readable, because it can only perform its function if it remains synchronized with the order of the rows in ImagePopHistory.csv.

The < **ImagePopOperations.txt** > now contains a list of every operation name that has been passed to the run command, in the order of first appearance. Since it was created by the write\_sealed\_text function of ImagePopLib, it also contains an ImagePopConfig.py specified header that remains separate from any data that follows. Each row contains the string name of every operation the user has passed to impop run if- and only if- it is understood by ImagePopLib. Adding the name of your own python files to the config file, in the manner that png.py and starfind.py already have been, will allow you to extend the range of understood operations.

The < **ImagePopHistory.csv** > now contains an excel readable table with one more column than the number of data lines in ImagePopOprations.txt and the same number of rows as the number of data lines in ImagePopIndex.txt. Each cell ought to contain a 24 character timestamp generated by python’s time module. The timestamps in the first column always correspond to the file modification time that impop was last aware of for each input file. The timestamp in any other cell indicates the file modification time for the output generated by calling that column’s corresponding run argument on that row’s corresponding file. These can be blank (full of 24 zeros) if there does not exist an output file named properly in the column’s corresponding run argument output directory, or if the input is modified/missing- which will flag the output to be generated in impop’s next session utilizing each run argument. For example, one row may be entirely zeros (except in the first column- corresponding to the input), but only the cells in columns corresponding to the currently active run argument will potentially be updated; the remaining cells remain zero until their column’s argument is run and the input file remains accessible.

- Appendix -

**Authors**:

Lowell Everest Schudel, email: [les2185@columbia.edu](mailto:les2185@columbia.edu),

Sol Weatherford Courtney email: [swc2124@columbia.edu](mailto:swc2124@columbia.edu)

**Github**:

[github.com/Schudel888/ImagePop](http://github.com/Schudel888/ImagePop)

**Youtube**:

<https://www.youtube.com/watch?v=k1fDdByZWY8>

**Course and Term Information**:

ASTRW3646x – Observational Astronomy – Fall 2015

Professor David Schiminovich, Jose Zorrilla Matilla

**Operating System used for demonstration**:

Mac OS X Yosemite version 10.10.5

**Python version used in demonstration**:

Python 2.7

**Python Modules required for ImagePop.py**

sys, time, csv, collections, numpy, astropy, itertools, matplotlib, os

**Files Generated by ImagePop:**

**In a Parent Directory**

ImagePopReadme.txt

ImagePopConfig.py

ImagePopLog.txt

**In a Target Directory**

ImagePopInit.txt

ImagePopOperations.txt

ImagePopHistory.csv

ImagePopIndex.txt

.ImagePopLock

<OPERATION\_NAME\_DIR>

**Demonstration Operations**:

<sf> Point source identification routine in starfind.py

<png> Converts a fits file’s 0th HDU to a png image using imshow, in png.py

<png\_contour> Converts a fits file’s 0th HDU to a png image using contour, in png.py

**Mock Data-Set File Names**:

OSMOS\_N123\_2p4

OSMOS\_N1\_2p4-transitData

OSMOS\_N1\_2p4\_solsOrion

**Quick Commands for demonstration:**

<Terminal 1>

cd

git clone <https://github.com/Schudel888/ImagePop.git>

cd ImagePop

mkdir ImagePopTarget

cd

sudo nano .profile

alias impop=’python ~/ImagePop/ImagePop.py

impop init ImagePopTarget

cd ~/ImagePop

impop run ImagePopTarget

sudo nano ImagePopConfig.py

WAIT\_INTERVAL and LEASH\_LENGTH

<Restart Terminal>

<Open 3 Terminals>

impop run ImagePopTarget sf

<Terminal 2>

cd ~/ImagePop

impop init ImagePopTarget/sf

impop run ImagePopTarget/sf png

<Terminal 3>

cd ~/ImagePop

MockDataSender