**A Python Implementation of**

**Practical Parallel Image Processing and Classification**

From the ground up

Sol Courtney, Lowell Schudel

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

**1**

**- Introduction -**

This paper has been written for the purpose of fulfilling the requirement of the fall 2015 Observational Astronomy (course number) final project. This paper will describe in detail the inner workings of a newly developed image processing python program named Image\_pop.py and its companion library Image\_pop\_lib. 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.

The program ImagePop.py its currently developed and supported by us, the authors of this paper, Lowell Everest Schudel and Sol Weatherford Courtney, both students of 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 under a telescope.

- 1.2 Motivations -

This project is driven by the notion that our data today already hold the answers to questions that we yet not know to ask. We believe that the 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. 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 “step-zero” software based automated data sorting and manipulation will realize a benefit to virtually any that elect its utilization. We hoped then and still do now that by providing this application we could make a small but positive contribution to the much larger ongoing process of scientific exploration; scientific exploration being the process to which we are both comfortably beholden.

- 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 and specific processes witch are typically performed on the majority of .fits data or only on a particular set. The user only needs to know the operations ahead of time to gain from ImagePop. By selecting the “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 otherwise have 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 record-keeping function of ImagePop. All functions and operation regardless of complexity, will be chronologically recorded and can be, at any time, referenced by the user at will. This is an important consolidation for consistency. 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 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 operation the user may have into its library without fuss. The user can easily run simultaneous instances of the program on a single data set. Above all it is absolutely open and free to all.

For these reasons stated above and for several others not yet mentioned, we feel confident in ImagePops overall performable efficacy, its value to the user and its quality as an example in exercise of contributive effort for the betterment of others. While ImagePop is not complex or deeply innovative, its plausible utility and the amount of effort required to develop it, is why, for this assignment, we decided to make it into something real and for everyone.

**2**

* **G-eneral Descriptions & Content Overview -**

All of the required items needed to preform this demonstration are available from the GitHub Repository (OUR GIT) and should be downloaded together if reproducing this demonstration is intended. The system requirements for this demonstration include an operating work station equipped with an implementation of Python 2.7, all Python modules listed in the appendix, a program for viewing “fits” files if the user wishes to inspect the output and a working internet connection.

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.

- 2.1 Software -

ImagePop.py is the only program the user needs to run. By selecting arguments, the user can achieve every possible function provided by ImagePop. 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. Image Pop can only run if an initialized directory exits and the PATH to this directory has been supplied to ImagePop.

**Note:** *The target directory contains all of the user input and is the directory that must be initialized.*

After a directory has been initialized the 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 directory are all itemized and defined in the appendix and will be discussed later, as they become germane for the purpose of our demonstration.

User defined operation guidelines for conformity. \*\*\*

The operations, which ImagePop can perform on the data, are intended, but not required, to be listed within ImagePopLib.py. By following only a few simple conformal requirements, the user can add operations to the library without limit. The operations which ImagePop 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.

- 2.2 Hardware -

The goal of this project is to have a parallel application capable of utilizing multiple machines at once to achieve a single task. This application would be equipped to access and assess the available resources 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 suitable aspects. In the future we intend to develop the parallel version of our application, which we expect to be far more powerful and far more interesting.

The user only needs a basic workstation equipped with any of the available common operating systems and a working Python 2.7 package.

- 2.3 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 (NAME) 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.

The ‘fits’ file header will be the item ImagePop first engages with. Before operation are executed, all data contained in the header will be properly read, interpreted and written for each file the program encounters. We will not use spectrographic images nor will we use any image in which stars cannot be rendered as point sources. This collection of data we are using is then very simple and is of course, not 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, 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 much more.

We simply supply our initialized directory with the mock data and ImagePop goes to work just the same. This way we can intentionally tweak the incoming data, specifically the headers, for the purpose of testing ImagePops’ 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.

- 2.4 Demonstration -

The demonstration video will cover everything from the program download to reviewing the format of outputted data. The video is accessible on YouTube (MORE) and can be acquired by contacting either of this papers’ authors. While the demonstration is intended for grading, the content covered in the paper can easily, and is intended to, serve as an installation guide and basic user manual for future users.

**3**

**- Demonstrating Operability -**

This is the last and most crucial section of the paper. 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 or a hard copy of the required material. Again, all necessary material can be gained by contacting either of the authors or by cloning the repository. Your workstation needs to have a working Python 2.7 distribution and all Python Modules listed in the appendix installed and working. This is 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.

- 3.1 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 ImPopTest. Within this directory we can make and do everything we need.

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

mkdir ImPopTest

cd ImPopTest

mkdir MyTestDir

Now we can download the repository into the new top-level directory ImPopTest.

Run the following:

get clone ######

ls –a

If the download was successful you should see a variety of new files. Once the download is complete and all files are copied to the ImPopTest directory we need to add an alias to the local .bashrc. 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:

cd

sudo nano .bashrc

At the bottom of the file add the lines:

alias impop=’python /your/path/to/ImPopTest/ImagePop.py’

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

After exiting and saving, restart the terminal session and reenter the ImPopTest directory. At the beginning of the next terminal session you can verify the alias is working by running:

which impop

- 3.2 Initializing your Directory -

ImagePop can only run from an initialized directory. Initializing a directory allows ImagePop to operate, it provides the awareness of the application insofar as what operation have been called and to what files, it allows the user to access the user configuration file and it allows the user to add operations to the library. If the user attempts to ‘run’ ImagePop in an un-initialized directory, ImagePop will automatically revert to the ‘init’ function and act as if the user intended to initialize the directory. The next step then is to initialize the ImPopTest directory and confirm our command was successful.

**[STEP 2]:** From the ImPopTest directory, run the following:

impop -init

ImagePop will then ask for a directory to initialize or <enter> to initialize the current directory. Press enter to continue.

Once the directory has been successfully initialized, several new files will exist in the directory:

1. **ImagePopIndex.txt**

Ordered File List generated into target directory. This is not essential to the functions of the demonstration but is part of a expanded version of ImagePop not yet completed.

1. **ImagePopOperations.txt**

Ordered Command List generated into target directory

1. **ImagePopTable.npy**

Table of Time Stamps generated into target directory

1. **ImagePopConfig.py**

Will be generated into runtime directory, if not user made. Can be supplied in the runtime directory.

Unless specified by the user, ImagePop will source the ImagePopConfig.py file in the top-most directory. In our case, this is the ImPopTest directory. Now that the directory has been initialized we are ready to execute our test run on the test data.

- 3.4 Running ImagePop.py -

Although there is no data in the test directory, we can start ImagePop 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, ImagePop is alert and operational, waiting to discover new files saved to its designated directory.

The command to run ImagePop will accept arguments. The arguments will reference individual operations that the user wishes the application to preform once on all data saved to the designated directory during the time ImagePop is alive. 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.

**[STEP 3]:** From the terminal, run the following command:

impop -run -t=0 -sf -png1

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

1. **<-run>** - is now where <-init> was in our previous command. In the second position either <-init> or <-run> must be present, always.
2. **<-t=0>** - is the third argument and when present it sets the time interval ImagePop will wait before checking the directory for new files. This is one of the parameters the user can set in the ImagePopConfig.py file. We have set it equal to zero because we are testing and therefor prefer ImagePop to check the directory for new files every second.
3. **<-sf >** - is a supplied operation. It will determine point sources within a specific value range.
4. **<-png1>** - will make a small “png” thumbnail image for each identified file. This is only for convenience.

Now ImagePop is alive and looking to the target directory for newly saved image files. As soon as we supply the test data, ImagePop will begin operating and outputting to the target directory. ImagePop will remain active until a specified amount of time has passed with no new files being saved to the target directory or until the user kills the terminal.

3.5 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 <NAME>. 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 MyTestDir so long as they both live in the same directory. If needed, the <NAME> script can be easily altered by the user

3.6 Understanding the Output

4

- Appendix -

1. Authors
2. Course and Term Information
3. Operating System used for demonstration: Mac OS X Yosemite version 10.10.5 (14F27)
4. Python version: 2.7
5. Python Modules required for ImagePop.py:
6. Files Generated by Initializing a Directory
7. Demonstration Operations
8. Mock Data-Set File Names