**The First of Three Steps**

**In Developing**

**A New Implementation of**

**Practical Parallel Image Processing and Classification Software**

**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. 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 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 and a working internet connection and a basic understanding of using 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 achieve every possible function provided to ImagePop 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 refered to as the “target-directory”. 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. 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.

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

The operations, which ImagePop 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 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.

- 2.3 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 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.

- 2.4 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.

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 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 wesite 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, specifically the headers, 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 (MORE) 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 -**

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 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 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 https://github.com/Schudel888/ImagePop.git

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

After which you should see the PATH to impop.

- 3.3 Initializing your Directory -

ImagePop 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 can only run if the target directory is an initialized directory. The “parent-directory” is the directory where ImagePop is being run from. We will be running ImagePop from the newly created ImPopTest directory. Initializing a directory allows ImagePop to operate within that directory, 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 MyTestDir directory and confirm our command was successful.

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

impop –init MyTestDir

ImagePop will then ask to initialize, press <enter> to initialize the directory.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. **ImagePopInit.txt**

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

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 The Configuration File -

The time interval ImagePop will wait before checking the directory for new files is the only parameter we must manually set in the file named **<ImagePopConfig.py>**. We need to set the line starting with “WAIT\_INTERVAL” equal to zero because we are only testing and therefor prefer ImagePop to check the target directory for new files every second.

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

cd /your/path/to/ImPopTest

sudo nano ImagePopConfig.py

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

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

The time interval is now set to one second between refreshing and we can move on to running ImagePop. Under normal operating conditions, a longer interval is desired for the purpose of conserving computational resources on the workstation.

- 3.5 Running ImagePop.py -

Although there is no data in the MyTestDir 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 target directory.

The command to run ImagePop will accept arguments. The arguments 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. Here we will be using several supplied operations.

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

impop -run -sf -png1

**<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 determine point sources within a specific value range.

**<-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.

3.6 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 MyTestDir 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 second terminal, run the following commands:

cd /path/to/ImPopTest

bash MockDataSender.sh /path/to/ImPopTest

You should now see the output messages of this bash script as it begins a periodic depositing of the mock data set into the MyTestDir directory. Now ImagePop will be able to detect this newly saved “fits” data in the 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. For example, if three arguments are passed to ImagePop, then the output will be in three separate directories all within the target-directory, each new output directory will be named according to the operation performed on the files it contains.

- 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)

**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**

**Files Generated by Initializing ImagePop in a Directory**

ImagePopReadme.txt

ImagePopConfig.py

**Files Generated by Running ImagePop**

**Demonstration Operations**:

Star Find

Png

**Output File Names**:

**Mock Data-Set File Names**:

OSMOS\_N123\_2p4

OSMOS\_N1\_2p4-transitData

OSMOS\_N1\_2p4\_solsOrion