A Compilation and Description of Scripts Written for the Optimization of Photometric Procedures

Emily Alyssa Ranquist

A senior thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Bachelor of Science

Dr. Denise Stephens, Advisor

Department of Physics and Astronomy

Brigham Young University

August 2013

ABSTRACT

A Compilation and Description of Scripts Written for the
Optimization of Photometric Procedures

Emily Alyssa Ranquist
Department of Physics and Astronomy
Bachelor of Science

Magnitudes are the quantitative measurements of a star's brightness. By analyzing a plot of the magnitude over time, known as a light curve, one can detect objects with fluctuating light output, such as variable stars, eclipsing binaries, and stars with transiting planets. To obtain magnitudes, we must go through a time consuming process called photometry on multiple images for every star we wish to plot. Through the use of the command language and scripting tools in NOAO's Image Reduction and Analysis Facility (IRAF), we have been able to develop a program called brightER that greatly reduces the amount of time spent on photometric procedures. BrightER is a compilation of multiple scripts, each designed to automate a specific part of the process. Using this program, we have been able to reproduce light curves in less than an hour that originally took days to create. The ease and efficiency of brightER allows us to focus less on data acquisition and more on the results.

Keywords: brightER, IRAF, light curve, magnitude, photometry

ACKNOWLEDGMENTS

This research was funded by Brigham Young University and could not have been completed without the professors and students in the Department of Physics and Astronomy. I would like to thank my advisor, Dr. Denise Stephens, for all of her help and advice during the research and writing process, as well as for teaching me the basics of astronomy. I would also like to give a special thanks to Joseph Rawlins, whose ideas inspired me to start this project.

A big thanks goes to Dr. Eric Hintz, Ben Rose, and Thayne McCombs for writing portions of the code used in brightER. Likewise, I would like to thank those who tested and helped improve it: Emily Stoker, Clément Gaillard, Tess Larson, and Carla June Carroll.

Lastly, I would like to thank my husband, Drake Ranquist, for all of his support and encouragement while writing this thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Light is crucial to the human ability to see. Our perception of the brightness of an object depends entirely upon the number of photons that enter the eye. Photometry is based upon this simple idea. Through “photon-counting,” astronomers are able to determine a quantity for the brightness of a stellar object, known as its magnitude. This value is used to determine distances, sizes, and characteristics of these objects.

Hipparchus (ca. 190-120 BC) was the first to create a classification for the magnitudes of stars. He classified the brightest stars as the first magnitude and dimmest as sixth magnitude (Ryden & Peterson 2009). This classification is still somewhat in effect; magnitudes increase in number as they decrease in brightness. Pierre Bouguer (1698-1758) was the first astronomer to devise a quantitative method for obtaining magnitudes. He did so by comparing the brightness of celestial objects to that of a standard candle flame (Sterken & Manfroid 1992). The same comparative method is used in photometry today, but the “standard candles” are stars with a known magnitude and distance.

Because of the limitation in human eyesight and the increase in technology, astronomers now obtain magnitudes through the use of telescopes equipped with a charge coupled device (CCD). Although CCDs have only been in use for the last 30 years, their contribution to photometry has been greatly significant. Their advantage is the ability to store light counts digitally from each individual pixel, which means an astronomer can choose which pixels to ignore when obtaining magnitudes (Howell 2011). CCDs opened the door for digital photometry and photometry packages.

One such photometry package was developed by a group at National Optical Astronomy Observatories (NOAO) called the Image Reduction and Analysis Facility (IRAF). This software system is capable of performing photometry for a selected group of pixel coordinates on a star field image, otherwise known as a light frame (NOAO Accessed June 17, 2013). IRAF is run via a terminal window in either Macintosh or Linux operating systems. It works simultaneously with SAOimage DS9, an imaging and data visualization program written by the Smithsonian Astrophysical Observatory (SAO) that allows the user to visually select stars and save the corresponding coordinates (SAO Accessed June 17, 2013). These programs, along with others, have become the standard method for modern photometry.

1.2 Purposes of Photometry in Astronomical Research

Photometry is a common method astronomers use to detect objects that have variable light output with time such as stars with transiting planets, variable stars, and high mass x-ray binaries. This is accomplished by performing photometry on a specific star field for multiple closely spaced intervals in time. The fluctuations in magnitude throughout this time period shed light on the nature of the observed object. For example, one can detect an extra-solar planet by observing the change in brightness of a star during a planetary transit. The brightness of the star decreases slightly as the planet begins to eclipse (ingress) and returns to its original brightness at the end

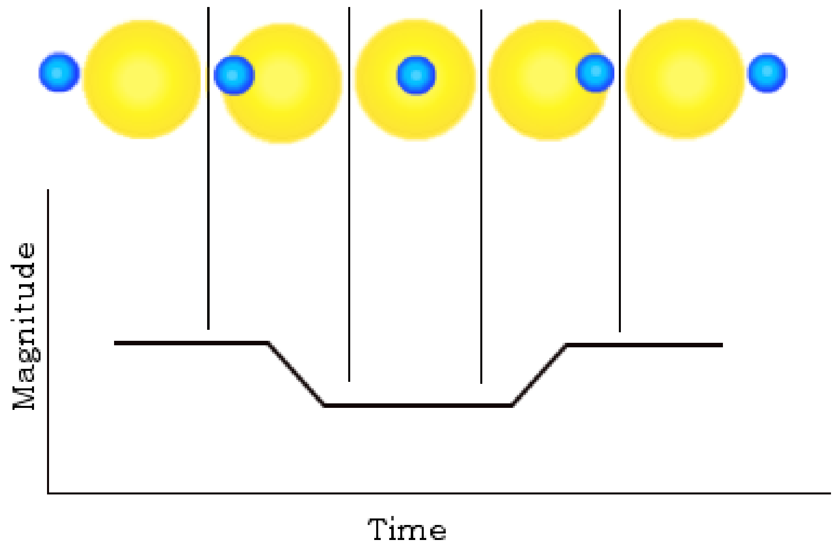


Figure 1.1 A diagram of a transiting extra-solar planet. As the planet begins its transit (ingress), the magnitude from the star decreases slightly. At the end of the transit (egress), the magnitude increases back to its original value.

of the eclipse (egress) (Foster 2010). A plot of the magnitude with respect to time is known as a light curve. One is able to classify an object by analyzing the depth and duration of this curve (see Figures 1.1, 1.2, and 1.3).

1.3 Procedures to Create a Light Curve

Although photometry is simple in concept, it is complicated in execution. There are multiple photometric procedures involved in creating a light curve. This section gives a brief description of each.

The first and most important step is acquiring the desired light frames from an appropriate telescope. Typically, there are 50-300 frames taken throughout the course of a night on one star field. These frames must be processed to remove any gradient in brightness caused by the equipment or the atmosphere, which greatly increases the signal-to-noise ratio (see Figure 1.4) . To do this, one

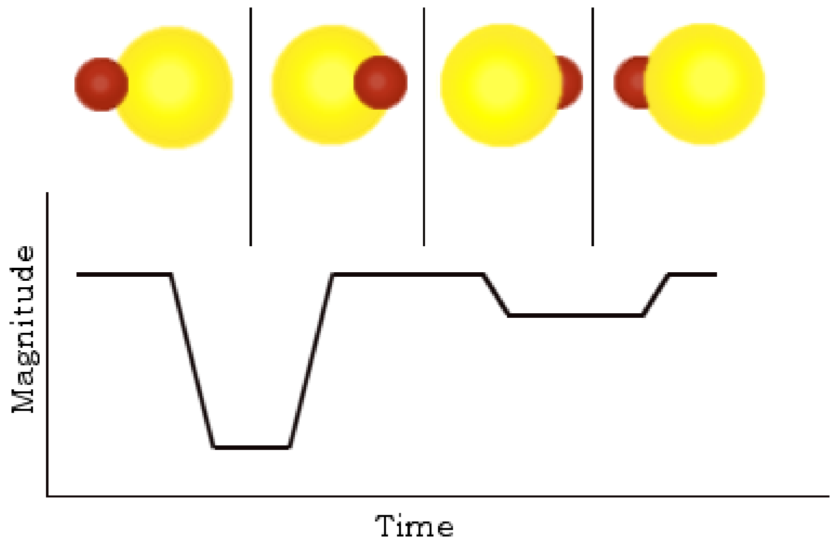


Figure 1.2 A diagram of a binary system. Both stars contribute to the total magnitude. The drops in magnitude occur when the smaller star eclipses and is eclipsed by the larger. Because the larger star outputs more light, there is a smaller drop when it covers the smaller star.

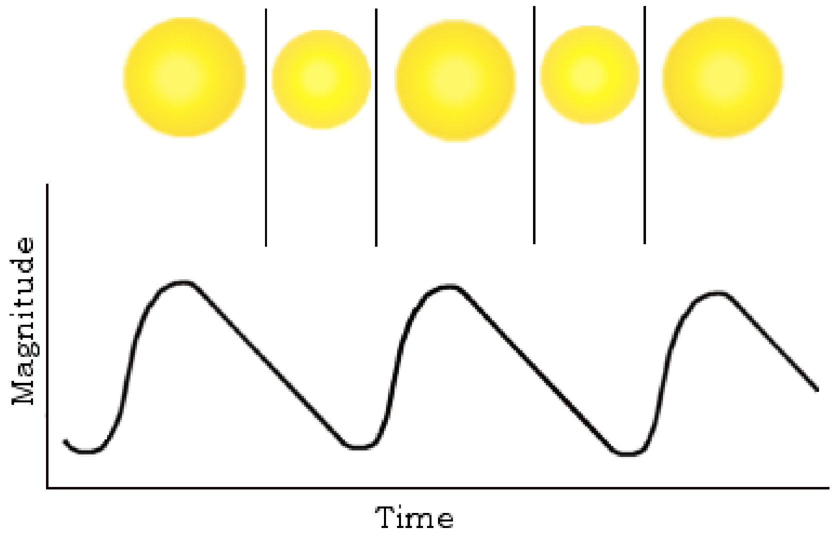


Figure 1.3 A diagram of a variable star. As the star pulsates in size, its magnitude increases and decreases accordingly. This results in an oscillating light curve.

must calibrate the raw frames using bias, dark, and flat frames. This process is explained further in section 2.1.

The next step is to create a coordinate file from selecting stars in the newly calibrated frames using DS9. In relative photometry, the average magnitude of a set of “comparison stars” is subtracted from the magnitude of the target object. For this reason, photometry is typically done on multiple stars in the field. If it is a crowded field, this can mean selecting 200-600 stars. The user must then save the selected stars as a coordinate file.

It is a good idea to view and remove unusable frames before running the photometry package in IRAF. Occasionally there are streaks of cosmic rays, oversaturated stars, or other problems with a light frame that cause errors during photometry (see Figure 1.5). The telescope can also slip while tracking the sky and double images of the stars can be produced.

After these steps, it is possible to retrieve magnitudes using IRAF’s `phot` command. This is located in the `apphot` package under the `noao` and subsequent `digiphot` packages. The user must specify each frame he or she would like to phot as well as specify the coordinate file previously created. The `phot` command outputs a magnitude file for each frame. This file includes a list of details such as the user, date, time, and image name, as well as the magnitude, errors, and Heliocentric Julian Date (HJD) of every coordinate listed in the coordinate file.

To create a light curve for a specific object, the user needs the magnitudes that correspond to the object’s location. Because magnitude files contain the magnitudes for multiple locations, it is important to know the exact coordinates of the target object. One must extract the magnitudes and corresponding HJDs for these specific coordinates from every file. This information can then be plotted through the use of any graphing software.

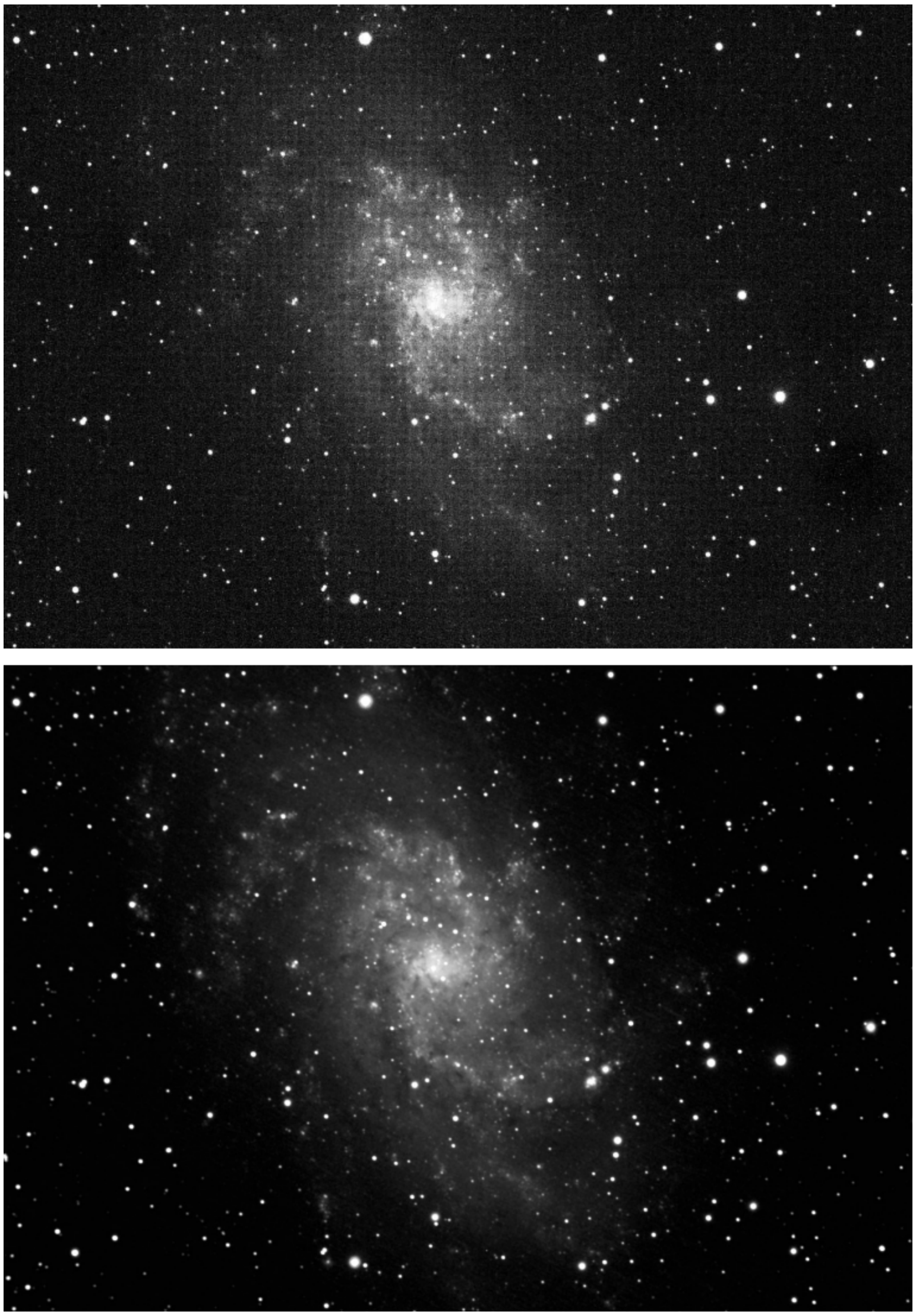
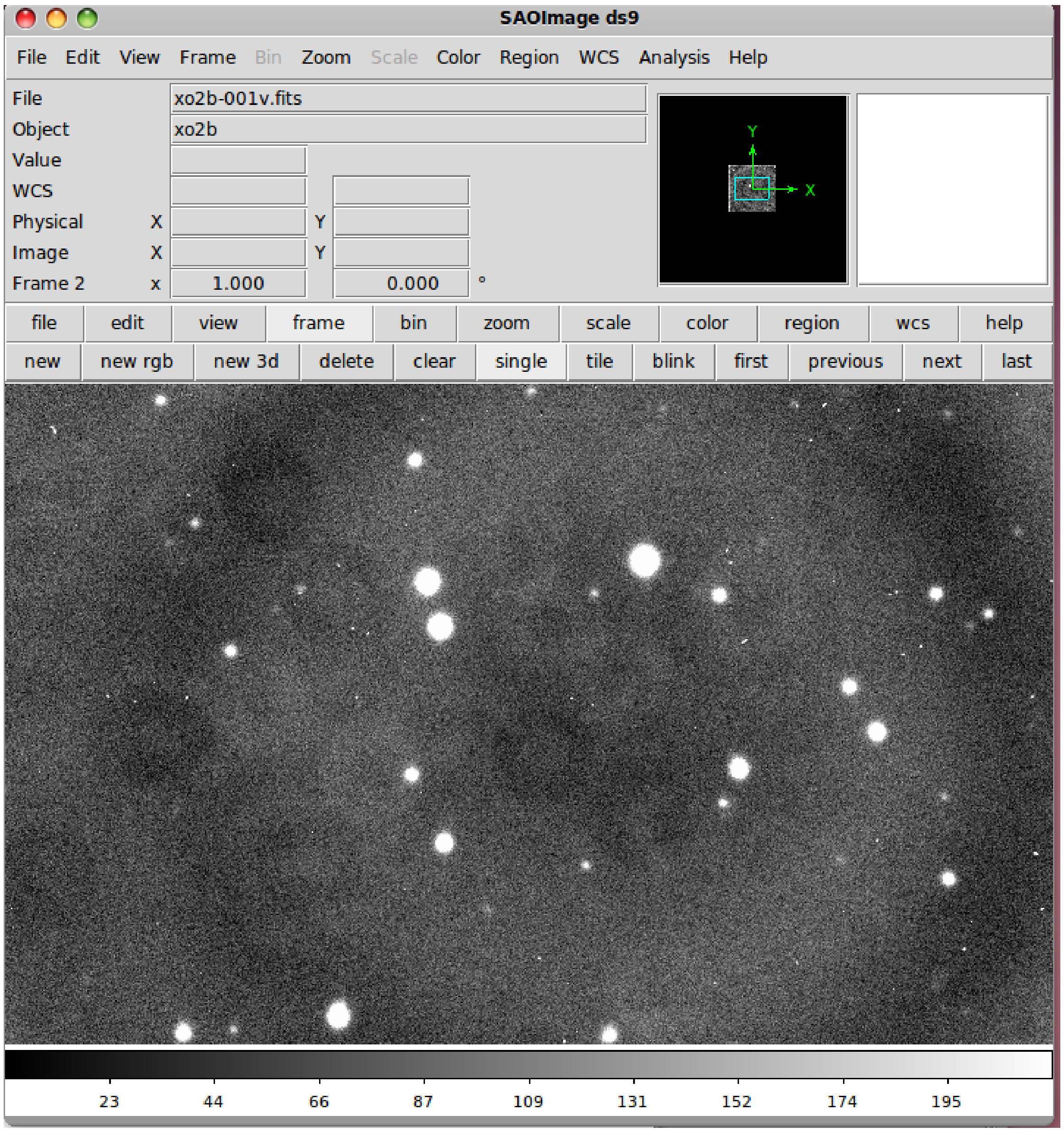


Figure 1.4 Both of these images show M33, the Triangulum Galaxy. The first is a raw frame taken with a 4 minute exposure using the Orion StarShoot Pro Deep Space CCD Color Imaging Camera. The second shows this same image after calibration.
Credit: Joe Roberts Astrophotography


SAOImage ds9
File Edit View Frame Bin Zoom Scale Color Region WCS Analysis Help
file edit view frame bin zoom scale color region wcs help
new new rgb new 3d delete clear single tile blink first previous next last
Y
X
23 44 66 87 109 131 152 174 195

File xo2b-001v.fits
Object xo2b
Value
WCS
Physical X Y
Image X Y
Frame 2 x 1.000 0.000 °

Figure 1.5 An unusable light frame of the XO-2b star field displayed in DS9. This image should not be used due to the non-uniform brightness of the sky background, which would likely give inaccurate magnitudes during photometry.

1.4 Compilation of Scripts known as brightER

The aforementioned procedures are time-consuming and tedious. Plotting a light curve is rarely accomplished in one sitting because it can take weeks to acquire the data to do so. In an effort to reduce the number of manual tasks required by the user and to increase speed and efficiency, the astronomy students at Brigham Young University developed a series of computer scripts that automate much of the photometric process. These scripts are each called from a master script that serves as a compact, user-friendly program. This program is known as brightER.

All of the scripts called by brightER, except two, are written in a custom IRAF programming language known simply as "command language." The advantage of this is that the scripts are run directly inside of IRAF. Because brightER takes advantage of many of the tasks inherent in IRAF, such as its `apphot` package, command language is the optimal coding tool.

The master script in the brightER package is essentially a series of if statements. It asks the user if he or she would like to do a certain task and waits for input. The input, either a yes or no, determines whether brightER calls a script specific to that task or moves on to the next question. Each of these scripts can also be run individually. Thus, if users wish to do one task only, they can call the script designed for that solitary task. Examples of these tasks include creating a coordinate file, viewing and removing unusable frames, running IRAF's `phot` package on multiple frames, averaging and organizing magnitudes, and plotting.

The following is the list of scripts in the brightER package:

1. fit2fits
2. regfile
3. viewframes
4. nphoter

5. varstar

6. plotsup

Additionally, there is a script used for processing frames called rfprocess. This is not included with the brightER package at present; however, it is a useful script that will likely be incorporated into brightER at a future point in time.

Most of these scripts, including brightER, were written by me personally, but the credit for varstar and nightphot4, which inspired nphoter, goes to Ben Rose, Dr. Eric Hintz, and Thayne McCombs, and the credit for the original plotsup goes to Joseph Rawlins, though it has been heavily modified.

1.5 Overview

This thesis discusses brightER and its contribution to the photometric process. Through photometry, astronomers detect and classify a variety of celestial objects. Magnitudes can be extracted through the use of IRAF's `phot` command, but the multiple steps involved make it an arduous process. BrightER makes photometry much quicker and more efficient by automating many of these steps through the use of multiple scripts. The remainder of this dissertation describes these scripts along with providing an explanation of the key components in their codes. Chapter 2 focuses on the scripts that must be executed before running IRAF's `phot` command; chapter 3 describes the scripts dealing with data and analysis, and chapter 4 discusses brightER's overall results and future work. Instructions of how to properly run and troubleshoot brightER are included in the Appendix.

Chapter 2

Pre-Photometry Requirements

2.1 Processing Raw Frames

As was previously mentioned, raw frames must be calibrated using bias, dark, and flat frames. An image taken with zero second exposure should, in theory, give a CCD readout of zero. In reality, the image gives a non-zero CCD readout known as the bias (McLean 1997). Therefore, this image is referred to as the bias frame, also known as the zero frame.

Many CCDs have the problem of "dark current," which is a small amount of current produced when there are no photons entering the detector. To isolate the readout from the dark current, we take several images with the telescope shutter closed so no photons can enter. These are known as dark frames and are exposed for the same amount of time as the raw frames.

Lastly, we take frames at twilight before or after observation using the same filters and exposure times as the raw frames. These are called flat field frames because the brightness is essentially uniform. Using these frames, we can eliminate transmission problems from optics and filters, increase quantum efficiency, and remove any additional problems caused by the telescope (McLean 1997).

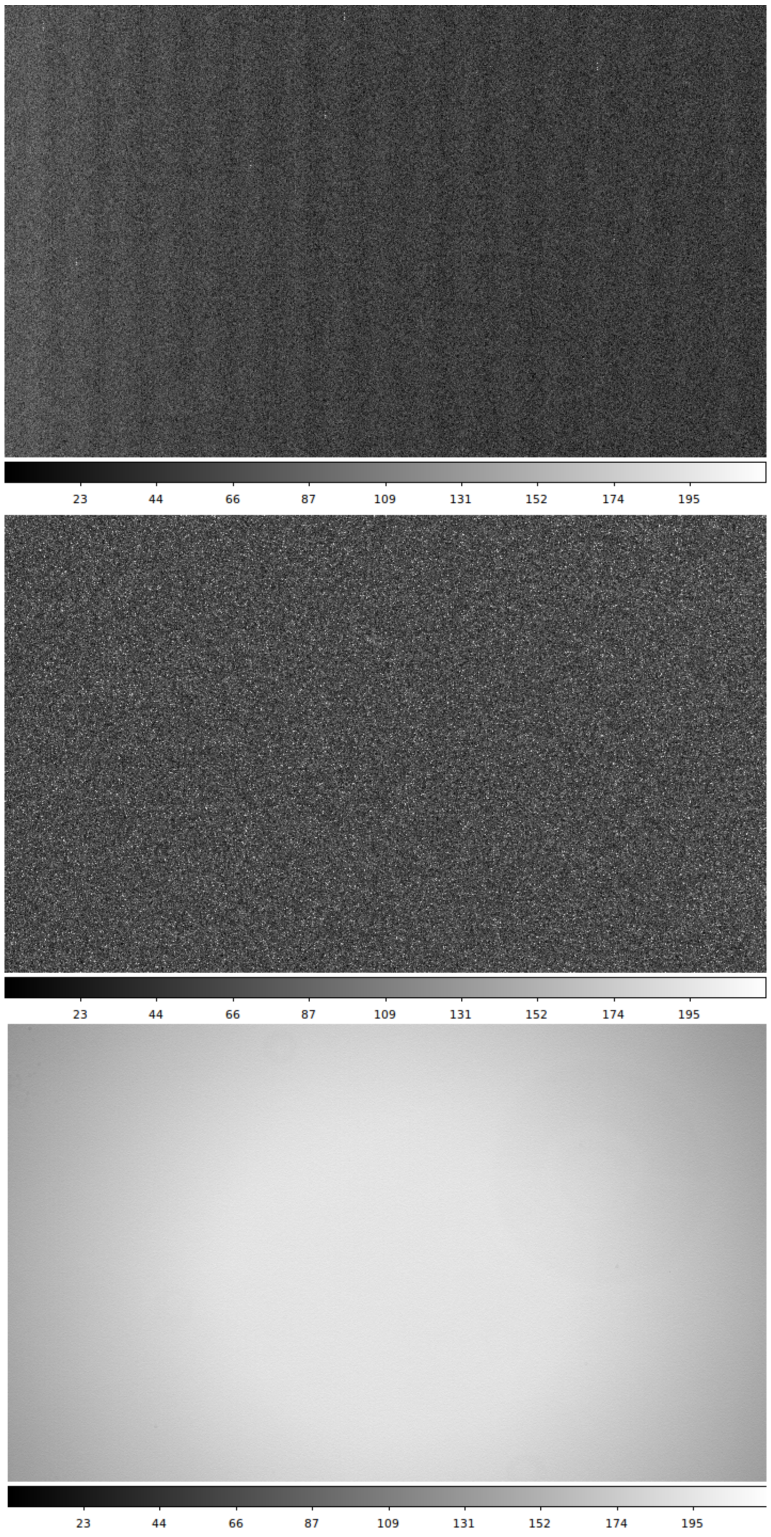


Figure 2.1 Master bias, dark, and flat frames, respectively. The flat and dark frames were each taken with 40 second exposures, and the flat frame was taken in the V filter. Each of these have been combined and processed.

Calibration refers to the removal of the bias and dark current, as well as the division of a normalized flat field, from the raw frames. To calibrate an image, one must first combine the bias frames to produce an averaged bias frame (see Figure 2.1). We subtract this master frame from the dark, flat, and raw frames to remove any bias within them. Following this procedure, we combine the dark frames that have a common integration time. These are subsequently subtracted from the flat and raw frames with that same integration time. This removes any dark current contribution. At this point, the flat frames are combined according to filter. For example, if raw data was taken in the I and V filters, one must combine all of the I flat frames and all of the V flat frames separately. These frames must then be normalized so that the average pixel value of the combined flat frame is one. Finally, the raw frames are divided by the normalized flat frame of the same filter. This results in a clean image that has retained all its pixel information. To put it simply,

$$Calibrated = \frac{(rawframes - Bias - Dark)}{Flat}$$

$$\textit{Bias= combined bias frames}$$

$$\textit{Dark=combined (dark frames} - \textit{Bias)}$$

$$\textit{Flat} = \textit{combined (flat frames} - \textit{Dark} - \textit{Bias) \& normalized.}$$

Rfprocess was developed to automate this procedure through the use of IRAF's `imcombine` and `ccdproc` tools. Before running rfprocess, the user must separate all of the frames according to integration time so that only frames with the same integration time can be combined and subtracted. Figure 2.2 shows lines 126-132 of this script, which is the entire image combination and subtraction process described prior. The following list describes each line's contribution to this process:

(127) Combines the bias frames and outputs an averaged bias frame named "Zero.fits"

(128) Subtracts "Zero.fits" from the dark frames.

(129) Combines the dark frames and outputs an averaged dark frame named "Dark.fits"

(130) Subtracts “Zero.fits” and “Dark.fits” from the flat frames.

(131) Combines the flat frames according to filter and outputs a file called “Flat(Filter Type).fits” For example, if working with frames taken in the V filter, it would output “FlatV.fits.” `Flatcombine` automatically normalizes this frame.

(132) Subtracts “Zero.fits” and “Dark.fits” from the object (raw) frames, and then divides by the normalized flat frame.

```
(126)    #Calibration Process
(127)    zerocombine (zero,output="Zero",ccdtype="zero")
(128)    ccdproc (images=dark,ccdtype="dark",zero="Zero.fits", dark="",
         flat="", zerocor=yes, darkcor=no, flatcor=no, fixpix=no,
         oversca=no,trim=no)
(129)    darkcombine (dark,output="Dark",ccdtype="dark")
(130)    ccdproc  (images=flat, ccdtype="flat", zero="Zero.fits",
         dark="Dark.fits",flat="",flatcor=no,zerocor=yes,darkcor=yes,fixpix=
         no,oversca=no,trim=no)
(131)    flatcombine (flat,output="Flat")
(132)    ccdproc(obj,ccdtype="object",flatcor=yes,zero="Zero.fits",dark="Dar
         k.fits",flat="Flat*.fits",fixpix=no,oversca=no,trim=no)
```

Figure 2.2 An excerpt from the code in rfprocess. This code shows the combination and subraction methods used to calibrate each image.

From line 130, we see that IRAF’s `ccdproc` tool is able to correct for flat, dark, and zero frames all at the same time. To calibrate images correctly, one must change the parameters *zerocor*, *darkcor*, and *flatcor* for the appropriate use. For example, in line 128, we wish to only subtract the bias frame from the dark frames. For this reason, *zerocor* is set to yes, whereas *flatcor* and *darkcor* are set to no. Everything shown in parentheses sets specific parameters before running the program, such as the image files to combine, output names, and names of the flat, dark, and zero frames to use when calibrating.

If one wishes to forgo running rfprocess, he or she can copy this code into the IRAF terminal. However, the first term after the parentheses in each line needs to be changed to the file name of the flat, dark, zero, or object frames. For example, if one's object frames were named "m330001.fits", "m330002.fits", etc., line 132 would read,

```
ccdproc("m33*.fits", ccdtype= "object", flatcor= yes, zero= "Zero.fits", dark=
"Dark.fits", flat= "Flat*.fits", fixpix=no, oversca=no, trim=no).
```

In addition to calibration, rfprocess edits each image header to include the title of the object, right ascension (RA), declination (DEC), and observatory ID. The observatory ID is a short title for an observatory that identifies its longitude, latitude, timezone, and full name. For major observatories, such as W.M. Keck , this information is stored within IRAF. Unfortunately, this is not the case for the three major observatories used at BYU: West Mountain, Orson Pratt, and ROVOR. The benefit of using rfprocess is that these values are stored within the code and the user needs only to type *wmo*, *esc*, or *rovor* to set the observatory parameters. It then sets the airmass and Julian date by using the RA and DEC previously added to the header. IRAF's `phot` command cannot run without this information.

Likewise, `phot` cannot run if the raw frames are stored with a FIT file extension. IRAF does not recognize this extension as an image, nor does DS9. To remedy this problem, both rfprocess and brightER call the script fit2fits. This script simply renames the file extension from FIT to fits. IRAF has a similar tool called `rfits`. Unfortunately, `rfits` does not rename the files, but creates new files with a generic root name. For example, the files "FlatV.FIT," "ObjectI.FIT," and "Zero.FIT" would become "Image001.fits," "Image002.fits," and "Image003.fits." This is particu-

```
(23)    #Changes .FIT to .fits
(24)    FIT = mktemp ("tmp$FIT")
(25)    sections ("*.FIT",option="fullname", > FIT)
(26)    list=FIT
(27)    while (fscan (list,img) !=EOF) {
(28)    length = strlen (img)
(29)    img2 = substr (img, 1, length-4)//".fits"
(30)    mv (img,img2)
(31)    }
```

Figure 2.3 An excerpt from the code in fit2fits. This code shows the algorithm used to rename any FIT files to fits files. FIT files cannot be displayed with DS9, which is an integral part of brightER.

larly problematic for images that need to remain separate and distinct. The advantage of fit2fits is its ability to change the file extension while keeping the root name intact.

Fit2fits is a very simple script, as shown in Figure 2.3, but it saves a lot of time and effort. The while loop in line 27 runs through each file with a FIT file extension and saves the name temporarily to the variable *img*. A substring then extracts every character from the name save the last four, which correspond to ".FIT." This leaves the user with the root name alone, which is subsequently concatenated with ".fits." We save the new image name as *img2* and use it to rename the original. DS9 will now display the images without any problems and the user can select stars to create a coordinate file.

2.2 Creating a Coordinate File for Specified Objects

If the user clicks on any areas of an image displayed in DS9, green markers will appear that can be adjusted in size and location. We use these markers to select stars and store their coordinates (see Figure 2.4). DS9 automatically names the coordinate file "ds9.reg," although this can be changed by the user. When saving the file, a dialogue box appears asking for the format and coordinate

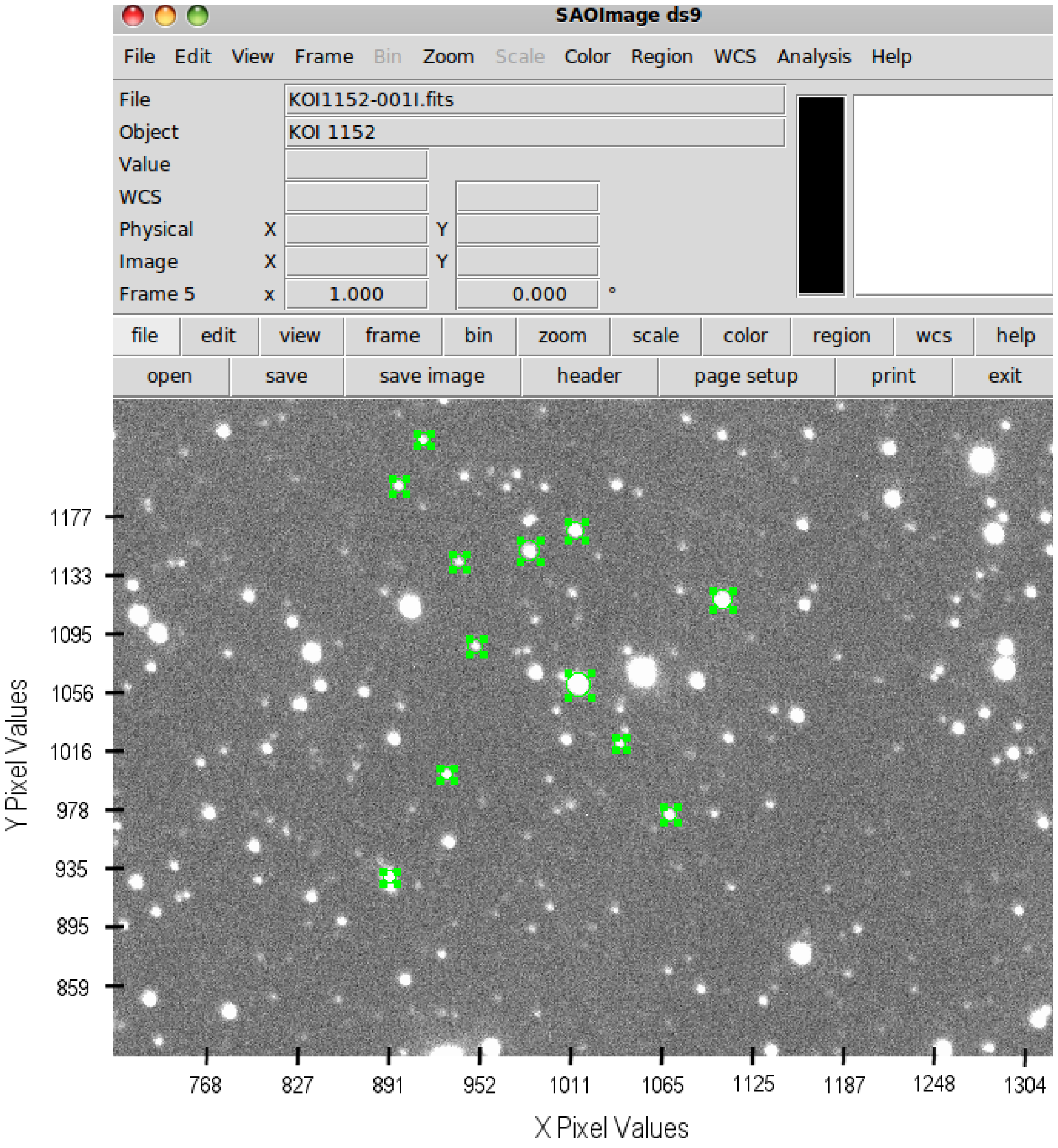
KOI-1152-I Filter		
Number	**X Pixel**	**Y Pixel**
1	887.452	926.219
2	1068.774	966.154
3	924.179	996.328
4	1036.546	1014.378
5	1009.202	1052.705
6	943.058	1079.468
7	1102.587	1105.298
8	932.228	1133.111
9	978.445	1136.43
10	1007.887	1153.924
11	892.961	1182.454
12	908.955	1210.973

Table 2.1 The coordinate file for the markers shown in Figure 2.4. Although the stars can shift throughout the frames, the coordinate number will remain the same. The user should know which number refers to the target object.

system. These need to be set to *xy* and *physical* in order to set up a Cartesian coordinate system. By opening "ds9.reg", one should see a list of x-y values based on pixel number (See Table 2.1). The user is able to load this file for any image displayed in DS9.

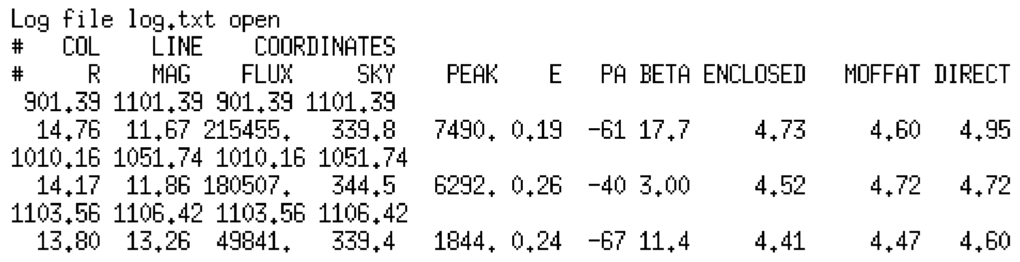
Creating a coordinate file takes time due to the multiple stars one generally selects in a crowded field. For example, the original coordinate file for Figure 2.4 contained 398 markers, each of which had to be adjusted to the approximate size of the stars they surrounded. IRAF has a command called `daofind` that one can use to avoid selecting stars manually. It uses the parameters *fwhmpsf* and *sigma* located in `datapars` in the `apphot` package to determine where stars are located in the frame and automatically marks and centers them. *Fwhmpsf* is the parameter denoting the average full width at half maximum (FWHM) of the point spread functions (PSF) of multiple stars. *Sigma* denotes the average standard deviation of the sky background.

These parameters must be set before running `daofind`. One way to obtain them is through IRAF's `imexamine` function. This will change the cursor to a blinking black circle. By hovering


SAOImage ds9
File Edit View Frame Bin Zoom Scale Color Region WCS Analysis Help
file edit view frame bin zoom scale color region wcs help
open save save image header page setup print exit
Y Pixel Values
1177
1133
1095
1056
1016
978
935
895
859
768
827
891
952
1011
1065
1125
1187
1248
1304
X Pixel Values

File KOI1152-001I.fits
Object KOI 1152
Value
WCS
Physical X Y
Image X Y
Frame 5 x 1.000 0.000 °

Figure 2.4 The KOI-1152 star field displayed in DS9. Shown are 12 green coordinate markers. These markers contain specific pixel coordinates listed in Table 2.1. For convenience, the pixels have been listed on the sides of the DS9 interface.



```
Log file log.txt open
#   COL     LINE    COORDINATES
#     R     MAG     FLUX      SKY     PEAK     E    PA BETA ENCLOSED   MOFFAT DIRECT
 901.39 1101.39 901.39 1101.39
  14.76   11.67 215455.    339.8    7490. 0.19  -61 17.7     4.73     4.60   4.95
1010.16 1051.74 1010.16 1051.74
  14.17   11.86 180507.    344.5    6292. 0.26  -40 3.00     4.52     4.72   4.72
1103.56 1106.42 1103.56 1106.42
  13.80   13.26  49841.    339.4    1844. 0.24  -67 11.4     4.41     4.47   4.60
```

Figure 2.5 The output from IRAF's imexamine tool. These values appear on the terminal window after hitting "A" on stars in the frame. A new line of values appears for each star selected.

the circle over a star and hitting A, a list of values will come up on the terminal screen. The value listed under "DIRECT" is the FWHM for that star (See Figure 2.5). Doing this for 8-12 different stars and recording the FWHM each time will provide the user with enough data to calculate a decent average for all of the stars in the image. Likewise, hitting M where no stars are located brings up a different list of values on the terminal screen. The value listed under "STDDEV" is the standard deviation for that location. Again, doing this 8-12 times at different points on the image background will give enough data to calculate an average. These two averages are the values the user needs to input into *fwhmpsf* and *sigma* before running `daofind`. BrightER reduces the number of steps the user must perform by calling `imexamine`, finding the averages, setting the parameters, and running `daofind` automatically through the script regfile.

Regfile begins by prompting the user for the name of the image, which is automatically displayed in DS9. It then calls `imexamine` and tells the user to choose values for the standard deviation by hitting M on the frame background 8-12 times. Regfile does not display the output from `imexamine`, but instead stores it as a log file where it extracts the standard deviation values and averages them. Afterwards, it asks the user to choose values for the FWHM by hitting A on 8-12 stars. Again, it stores the output and averages these values.

To extract the standard deviation and FWHM values from the `imexamine` log files, regfile uses IRAF's `columns` command. This command splits up the log files into multiple column files.

```
(60)    #Deletes the first 14 column files "col.1,col.2..."
(61)    #Column 15 is the fwhm file.
(62)    for (g=1; g <= 14; g+=1) {
(63)    del "col."//g
(64)    }
(65)    }
(66)    del log.txt
(67)
(68)    rename ("col.15", "tmp$fwhm")
```

Figure 2.6 An excerpt from the code in regfile. This shows the technique for extracting the FWHM and Standard Deviation values from the logfile output by IRAF's `imexamine` tool.

The standard deviations are in a file called "col.5" and the FWHM values are stored in "col.15." These two files are renamed to "tmp$stdev" and "tmp$fwhm," while the remaining column files are removed (see Figure 2.6).

Regfile then averages the values in these files by calculating them in the same way someone would with a calculator. The code in Figure 2.7 shows how this is done. The while loop in line 76 runs for every FWHM in the file "tmp$fwhm." Each time the while loop starts over, a new FWHM value is added onto the variable *j*. This variable continues to accrue values until the while loop reaches the end of the file. At that point, *j* is equal to the sum of all the FWHM values. The variable *h* shown in line 89 counts how many values there are by incrementing every time the loop restarts. Thus, the average FWHM is attained by dividing the final *j* by the final *h*. Regfile uses the same process to find the average standard deviation.

After storing these values into the correct parameters, regfile runs `daofind`. The output file gives the coordinates for stars everywhere in the image, including the edges. If a telescope does not track the stars across the course of the night, stars will shift location in every frame due to the Earth's rotation. This means that stars on the edge of the frame will disappear completely in later frames. Thus, any markers on these stars will eventually produce indefinite results. Joseph

```
list1="tmp$fwhm"
j=0;
h=0;

#Averages fwhm values from imexam
while (fscan (list1, line) != EOF) {
j=real(line)+real(j)
h+=1
}
avgfwhm=real(j)/h
```

Figure 2.7 A second excerpt from the code in regfile. The portion of this code shows how regfile calculates the average standard deviation and FWHM. These values set the parameters *fwhmpsf* and *sigma*, which are necessary to run IRAF's `daofind` tool.

Rawlins devised a way to solve this problem. He wrote a MATLAB script called daofindsort that used if statements to select only those coordinates below a specific threshold. He then used these coordinates to create a new coordinate file. Any coordinate above this maximum value would be ignored. Because IRAF cannot call a MATLAB script, we rewrote daofindsort into an IRAF script. Instead of using a maximum value, however, it prompts the user for a percentage of the field to view. For a crowded field, 35% is recommended.

Regfile saves the new coordinate file as "ds9.reg" in the same directory as the image. It urges the user to load the file, remove any bad coordinates, and re-save. Bad coordinates include markers on cosmic rays and oversaturated stars, as well as multiple markers on one star. It is also a good idea to re-mark the target object and re-save so that it will be the last coordinate listed in the file. This makes it easier to keep track of during photing and plotting. Afterwards, the user will have a coordinate file that will work with `phot`.

2.3 Viewing and Removing Unusable Frames

Just as cosmic rays and oversaturation cause problems for daofind, they can make entire frames unable to use. Skewed, streaked, dusty, oversaturated, or otherwise ruined images will cause `phot` to give errors and inaccurate data. As such, the user should view each frame before running `phot` by typing `display` into the IRAF terminal followed by the image name. This is a monotonous procedure because a user typically works with several hundred frames. If one does have a bad frame, he or she must manually remove it by deleting it from the folder containing the images.

Viewframes was written to accelerate this process by displaying multiple images at a time and removing those selected by the user. Although `display` only works for one image at a time, that image can be placed into 1 of 16 frames in DS9. Thus, viewframes runs `display` 16 times and places each new image into an unused frame. To view all 16 frames at the same time, the user must go to the *frame* tab and select *tile* in DS9 (see Figure 2.8).

At this point, viewframes asks the user if they would like to remove any of the frames. If so, the user simply types in the number of the frame displaying the bad image. This is found by hovering the cursor over the frame and looking just above the *file* tab, where the words “Frame” followed by the frame number are shown. Viewframes then moves the selected image into a folder named “badframes.” The script reruns through the entire process until the user quits or every image has been displayed. After running viewframes, brightER proceeds with photometry.

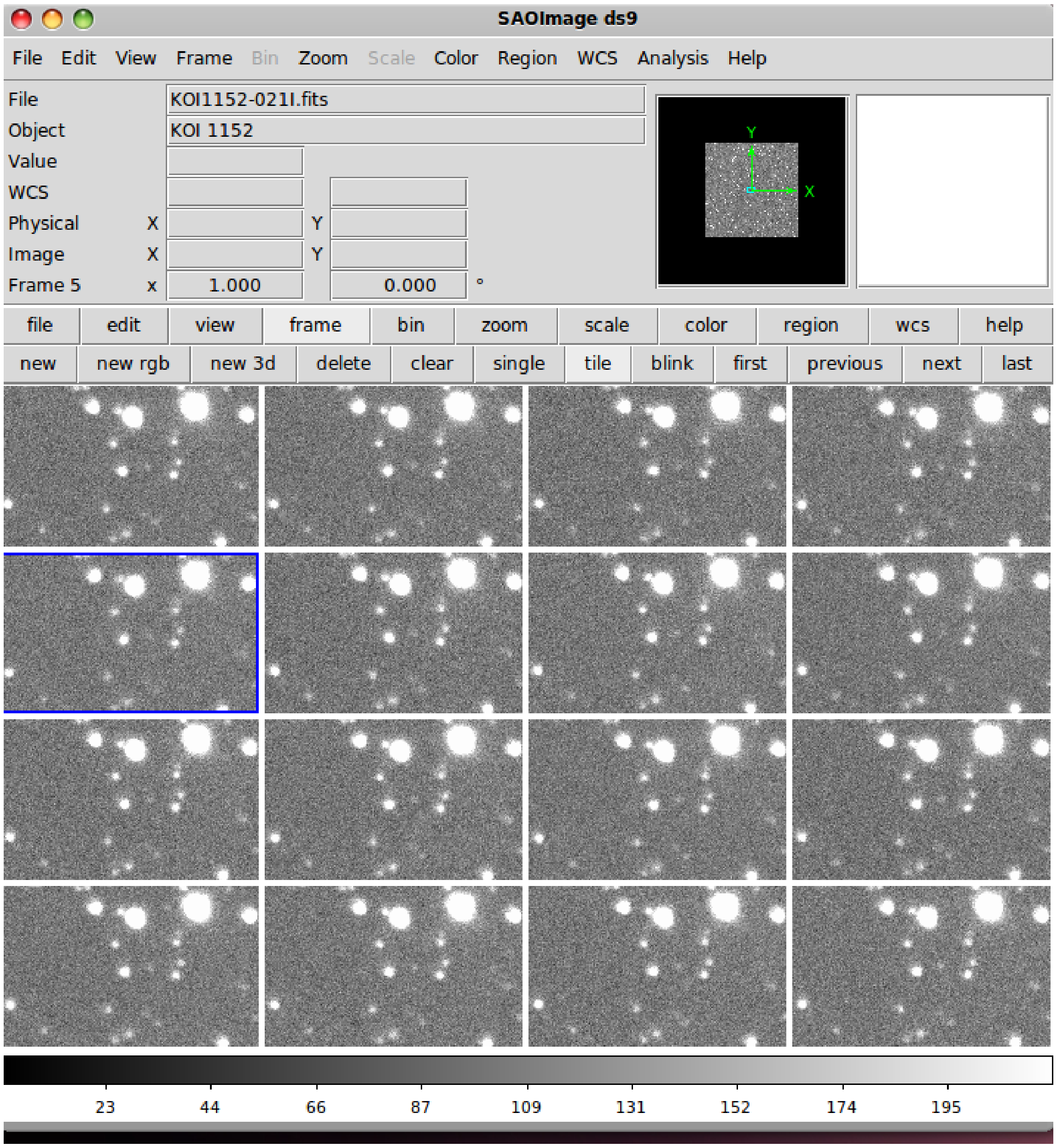

SAOImage ds9
File Edit View Frame Bin Zoom Scale Color Region WCS Analysis Help
File KOI1152-021I.fits
Object KOI 1152
Value
WCS
Physical X Y
Image X Y
Frame 5 x 1.000 0.000 °
Y
X
file edit view frame bin zoom scale color region wcs help
new new rgb new 3d delete clear single tile blink first previous next last
23 44 66 87 109 131 152 174 195

Figure 2.8 A 4 x 4 grid of images displayed in DS9's Tile Display. The frame number is displayed just above the file tab. The selected frame in this image is frame 5. This is the number prompted for by viewframes when the user wishes to remove a frame.

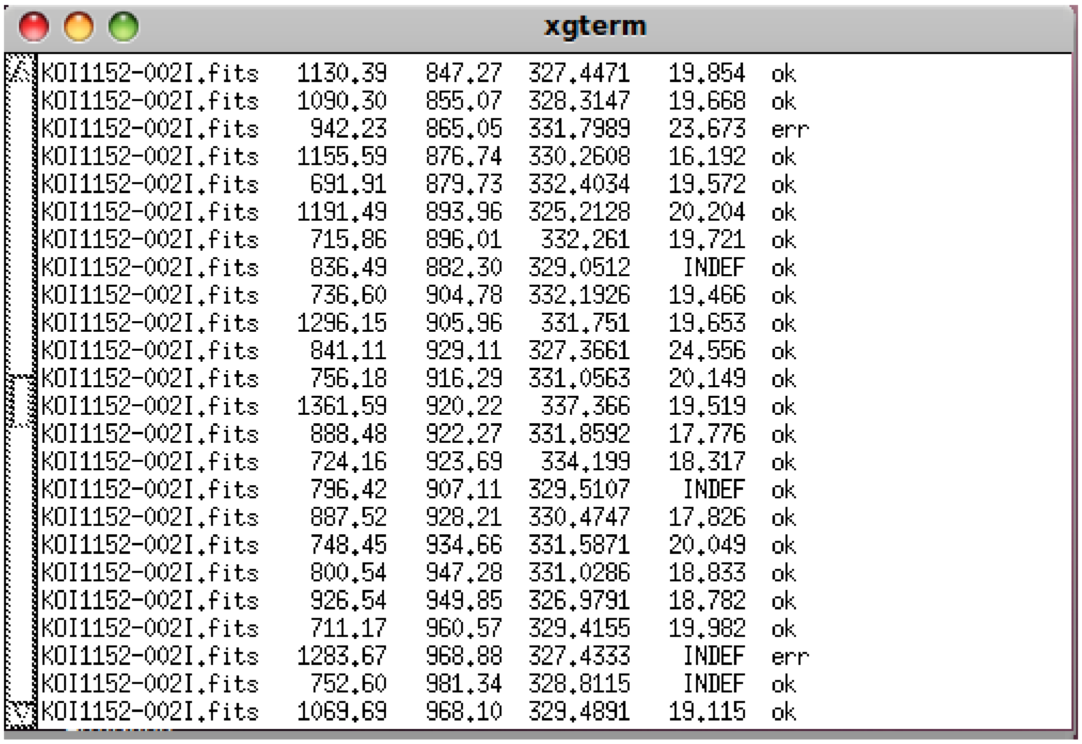
Chapter 3

Data Acquisition and Analysis

3.1 Performing Photometry for Multiple Light Frames

IRAF's `phot` command is able to phot multiple light frames at the same time; however, it only uses one coordinate file to do so. This becomes a problem if the stars are not in the same location in every frame. For small shifts, `phot` adjusts the coordinates to the center of the star and lists them in the magnitude file. As the frames progress, `phot` can no longer center the coordinates because the shift has become too large. For this reason, we typically phot each image individually, while shifting and re-saving the markers each time.

To make the process faster and more accurate, the script nightphot4 was written. It saved the user the time of typing in each image name and attempted to shift the coordinates automatically. It did this by extracting the centers from the previous magnitude file to overwrite the coordinate file. The idea was that shifts between two subsequent frames would be small. Occasionally, these shifts would still be too large for `phot` to adjust. As such, nightphot4 would ask the user each time if the coordinates were correctly aligned before moving on to a new frame. In addition to this, it would use IRAF's `psfmeasure` command to allow the user to remove any FWHM values that greatly



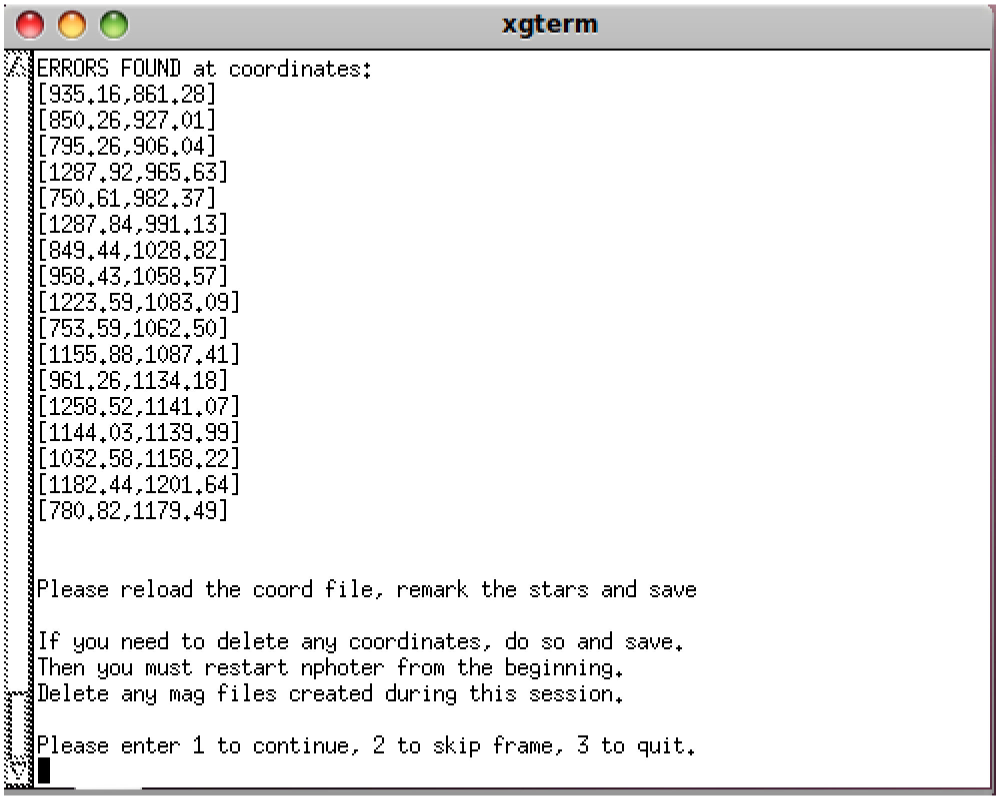
```
KOI1152-002I.fits   1130.39   847.27  327.4471   19.854  ok
KOI1152-002I.fits   1090.30   855.07  328.3147   19.668  ok
KOI1152-002I.fits    942.23   865.05  331.7989   23.673  err
KOI1152-002I.fits   1155.59   876.74  330.2608   16.192  ok
KOI1152-002I.fits    691.91   879.73  332.4034   19.572  ok
KOI1152-002I.fits   1191.49   893.96  325.2128   20.204  ok
KOI1152-002I.fits    715.86   896.01   332.261   19.721  ok
KOI1152-002I.fits    836.49   882.30  329.0512    INDEF  ok
KOI1152-002I.fits    736.60   904.78  332.1926   19.466  ok
KOI1152-002I.fits   1296.15   905.96   331.751   19.653  ok
KOI1152-002I.fits    841.11   929.11  327.3661   24.556  ok
KOI1152-002I.fits    756.18   916.29  331.0563   20.149  ok
KOI1152-002I.fits   1361.59   920.22   337.366   19.519  ok
KOI1152-002I.fits    888.48   922.27  331.8592   17.776  ok
KOI1152-002I.fits    724.16   923.69   334.199   18.317  ok
KOI1152-002I.fits    796.42   907.11  329.5107    INDEF  ok
KOI1152-002I.fits    887.52   928.21  330.4747   17.826  ok
KOI1152-002I.fits    748.45   934.66  331.5871   20.049  ok
KOI1152-002I.fits    800.54   947.28  331.0286   18.833  ok
KOI1152-002I.fits    926.54   949.85  326.9791   18.782  ok
KOI1152-002I.fits    711.17   960.57  329.4155   19.982  ok
KOI1152-002I.fits   1283.67   968.88  327.4333    INDEF  err
KOI1152-002I.fits    752.60   981.34  328.8115    INDEF  ok
KOI1152-002I.fits   1069.69   968.10  329.4891   19.115  ok
```

Figure 3.1 Output from IRAF's `phot` tool. The first two columns of values are the coordinates, followed by the sky values, and lastly the magnitudes of each star. The last column shows any errors that occured during the process.

offset the average. Nightphot4 used this average to set the *aperture* and *annulus* parameters, which help ensure that `phot` obtains an accurate magnitude for each star. After each time nightphot4 ran `phot`, a table would appear on the terminal screen, one column of which displayed either the words "ok" or "err" (See Figure 3.1). Nightphot4 would then ask if the user was satisfied with these results before continuing to the next image. If the last two columns ever displayed "INDEF" or "err," such as in this figure, the frame had to be re-photed. Although useful, nightphot4 still depended heavily on the user and, once started, could not stop except by a forced quit.

Because nightphot4 had many innovative ideas, it became the foundation for brightER's script, nphoter. This script uses the code from nightphot4 to start with the same prompts, shift coordinates automatically, and run `psfmeasure` to set the *aperture* and *annulus* parameters. However, it forgoes asking the user if the coordinates are centered. Likewise, it does not display a table and ask the user if they are satisfied with the results.

Instead of relying on the user to find errors, nphoter searches for them, displays their coordinates, and asks the user to fix them (See Figure 3.2). As soon as the specified coordinates have been shifted, nphoter re-phots and checks again for errors. If there are no errors found, nphoter will automatically continue onto the next image. Thus, nphoter is capable of photing every image without prompting the user at all. This allows the user to phot hundreds of frames with virtually no manual work. In addition to this, each time errors are found, the user has three options: fixing the error, skipping the frame, or quitting nphoter. If he or she chooses to quit, nphoter has an option at the start to continue from a previous session. Therefore, the user does not have to phot every frame in one sitting.



```
ERRORS FOUND at coordinates:
[935.16,861.28]
[850.26,927.01]
[795.26,906.04]
[1287.92,965.63]
[750.61,982.37]
[1287.84,991.13]
[849.44,1028.82]
[958.43,1058.57]
[1223.59,1083.09]
[753.59,1062.50]
[1155.88,1087.41]
[961.26,1134.18]
[1258.52,1141.07]
[1144.03,1139.99]
[1032.58,1158.22]
[1182.44,1201.64]
[780.82,1179.49]


Please reload the coord file, remark the stars and save

If you need to delete any coordinates, do so and save.
Then you must restart nphoter from the beginning.
Delete any mag files created during this session.

Please enter 1 to continue, 2 to skip frame, 3 to quit.
```

Figure 3.2 The output from nphoter. Typically, there will only be a few errors unless every coordinate needs to be shifted due to large jumps in the stars' locations between frames. The coordinates shown here were intentionally misaligned.

Figure 3.2 gives a warning to the user. If he or she needs to delete a coordinate rather than shift it, the entire nphoter session must be restarted. Any magnitude files obtained must be deleted. The reason for this is that one must always have the same number of coordinates from the beginning to the end. Although the coordinates themselves are realigned often, they retain their position in the list of coordinates. This coordinate number specifies a star. By removing a coordinate, each position changes and it becomes very difficult to know which coordinate number refers to which star. The same is true for adding coordinates after starting nphoter. Because the coordinate file is so crucial, nphoter makes a copy at the very beginning called “originalcoord.reg.”

Nphoter’s biggest advantage over nightphot4 is its ability to search for errors automatically. Figure 3.3 shows the code that makes this possible. It begins by putting the table shown in Figure 3.1 into a text file called “errors.txt.” The variable *list3* is then assigned to this file. Lines 149-154 show a while loop that scans through every column in the list line by line. Each column is assigned a name: *a ,b, c, d, mag*, or *err*. These names are used to extract a value from a particular column. Line 152 shows an if statement that searches through the columns *mag* and *err* (the last two columns) only. It essentially states that if the value in the magnitude column is the word “INDEF,” or the length of the word in the error column is 3 for “err,” then it will set the variable *q* to 1. This while loop also counts the total number of lines in the list by incrementing the value *j* in line 150 and setting the final value equal to the variable *m*.

If there are any errors (i.e. if *q* is equal to 1, not 0), it scans through the list one more time in line 162. For any place there is an error, it prints the columns named *b* and *c*, which are the x and y coordinates, onto the terminal screen (see line 166). For any place there is no error, the variable *n* increments by one. When *m* is equal to *n*, it means that the total number of lines in the list is equal to the number of places there are no errors. In other words, it means there are no errors in the list. When this is the case, the variable *q* is set to 0 meaning lines 157-171 are ignored and the program moves on to the next image. Nphoter will continue to run these lines of code until *m* is equal to *n*.

```
#Scans through error file output by phot
while (fscan (list3,a,b,c,d,mag,err) !=EOF) {
j=j+1
m=j
if(substr(mag,1,5)=="INDEF" || strlen(err)==3)
q=1
}
n=0

if(q !=0) {
clear
print("ERRORS FOUND at coordinates:")}

list3="errors.txt"
while (fscan (list3,a,b,c,d,mag,err) !=EOF) {
if(substr(mag,1,5)=="INDEF" || strlen(err)==3){

#Gives the x and y coordinates of errors.
print("[",b,",",c,"]")
}
else if(substr(mag,1,5) != "INDEF" && strlen(err)==2){
n=n+1
}
}

if(n == m){
q=0
```

Figure 3.3 An excerpt from the code in nphoter. This is the algorithm for finding and displaying error messages, such as the one shown in Figure 3.2.

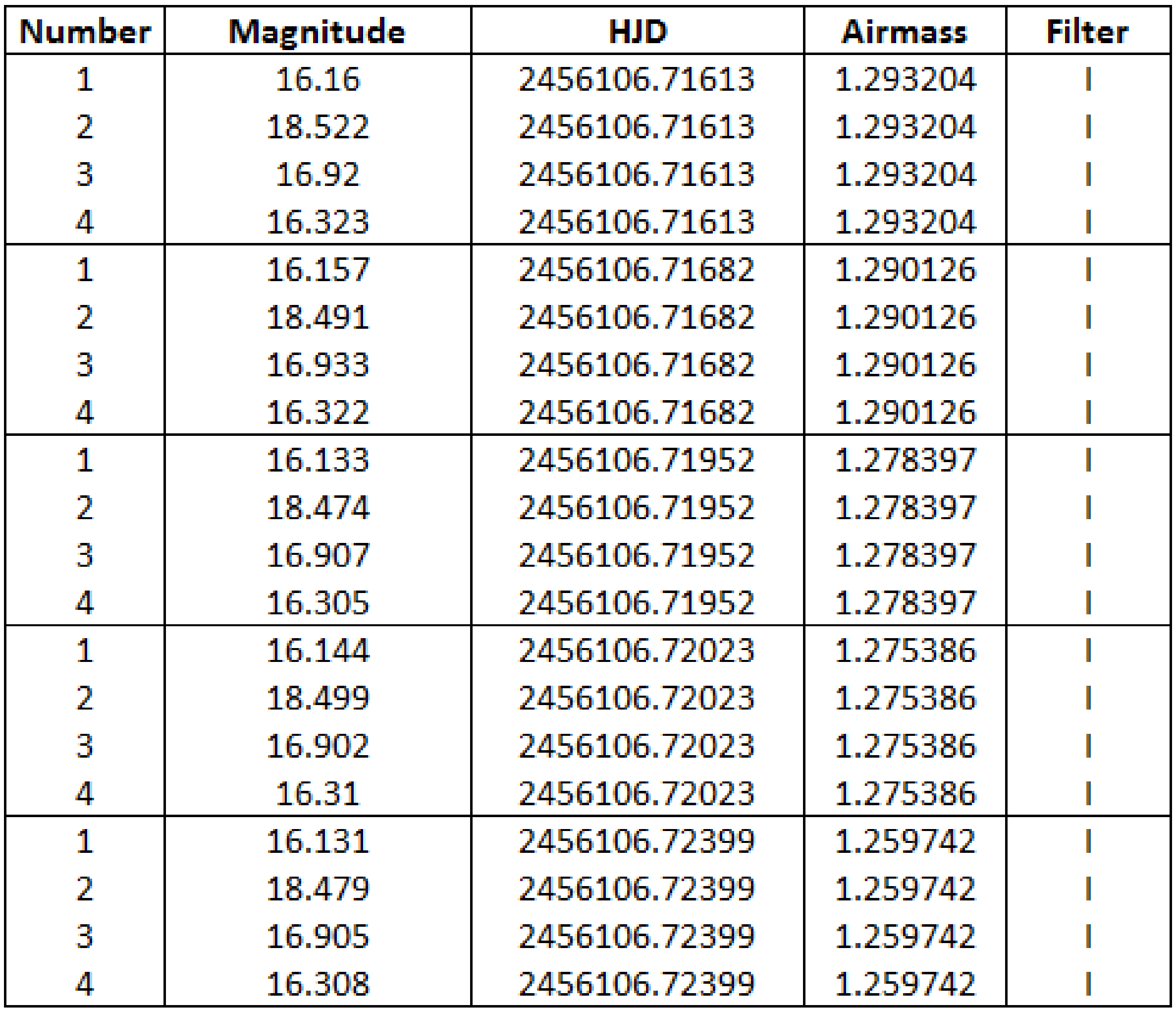
Every time there are errors, it will ask the user to fix them and then re-phot the image.

Currently, there are four different versions of nphoter. Unlike nightphot4, the nphoter version included in the brightER package ignores the outliers that offset the average FWHM. The second version prompts for a maximum FWHM and averages any values under that threshold. This only affects the *annulus* and *aperture* parameters. The annulus is 3 times the average, whereas the aperture is 1.15 times the average. Because the average FWHM changes for each frame, these parameters do as well. However, the third version of nphoter keeps a fixed aperture size, and the fourth has a fixed aperture as well as a maximum FWHM. These versions exist to determine which method of photometry provides the most accurate light curve.

3.2 Extracting and Organizing Magnitudes

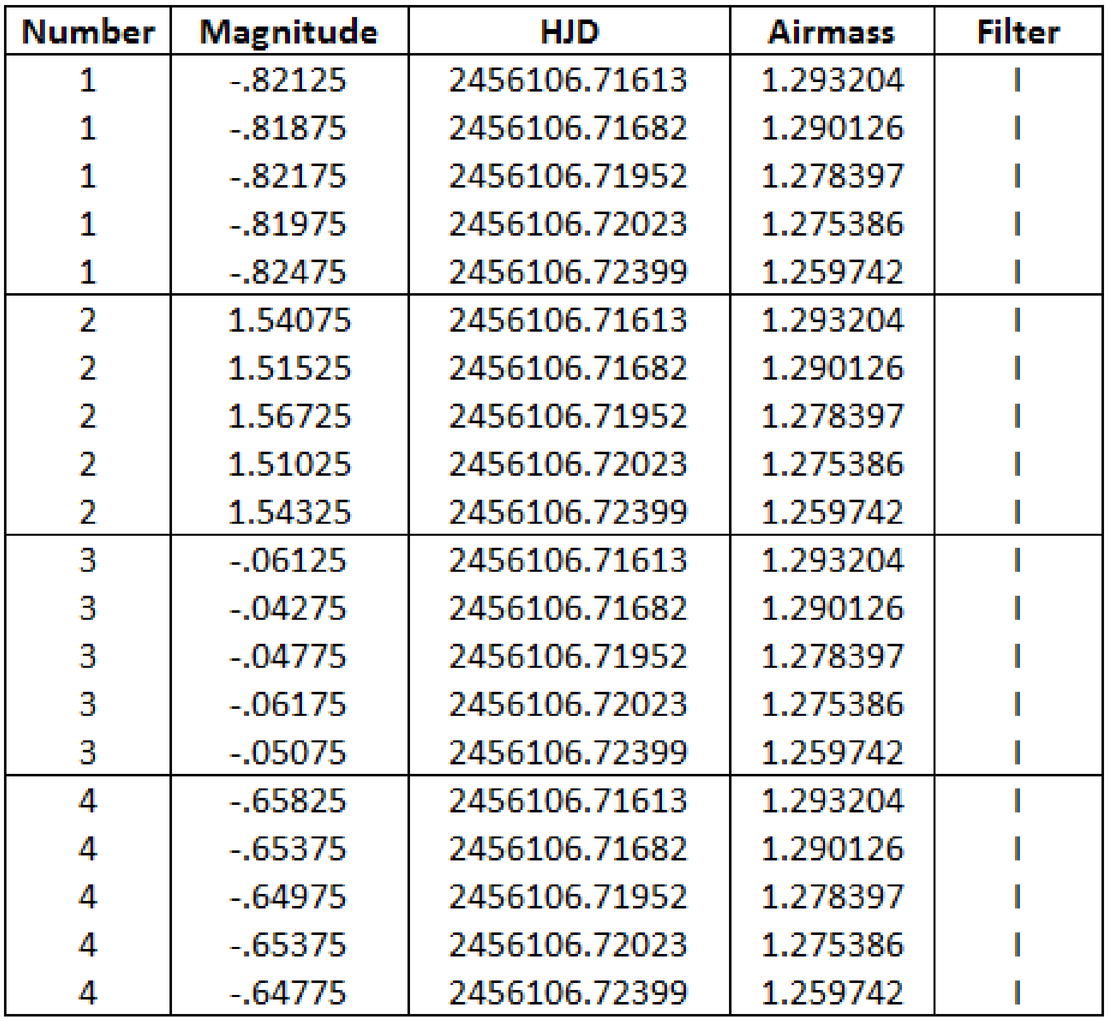
After successfully photing, nphoter creates an LST file that contains the coordinate number, magnitude, HJD, airmass, and filter from every image (see Table 3.1). The coordinate numbers represent the marked stars in a frame and restart each time a new frame's information begins. For example, in Table 3.1, there are 5 frames, each containing the same 4 stars. The HJD represents the time at which the frame was taken, thus each star in one frame will have the same HJD and a new frame will have a different HJD.

As mentioned previously, we do photometry on multiple stars in the field to use for comparison. Using these magnitudes, we create a constant value and subtract it from every magnitude within that frame. This constant value can be the average of all of the magnitudes in that frame, or it can be a super magnitude obtained by adding together every flux in the frame and converting it back into a magnitude. By subtracting this value from each magnitude in the frame, we create a reference point from which we can compare brightness. At this point, the user needs to organize the information according to the coordinate number (see Table 3.2). This puts all of the magnitudes



Number	Magnitude	HJD	Airmass	Filter
1	16.16	2456106.71613	1.293204	I
2	18.522	2456106.71613	1.293204	I
3	16.92	2456106.71613	1.293204	I
4	16.323	2456106.71613	1.293204	I
1	16.157	2456106.71682	1.290126	I
2	18.491	2456106.71682	1.290126	I
3	16.933	2456106.71682	1.290126	I
4	16.322	2456106.71682	1.290126	I
1	16.133	2456106.71952	1.278397	I
2	18.474	2456106.71952	1.278397	I
3	16.907	2456106.71952	1.278397	I
4	16.305	2456106.71952	1.278397	I
1	16.144	2456106.72023	1.275386	I
2	18.499	2456106.72023	1.275386	I
3	16.902	2456106.72023	1.275386	I
4	16.31	2456106.72023	1.275386	I
1	16.131	2456106.72399	1.259742	I
2	18.479	2456106.72399	1.259742	I
3	16.905	2456106.72399	1.259742	I
4	16.308	2456106.72399	1.259742	I

Table 3.1 The contents of the LST File created by nphoter. Boxes were added to delineate separate frames. The HJD for each frame remains constant while the magnitude changes for each star in the frame.



Number	Magnitude	HJD	Airmass	Filter
1	-.82125	2456106.71613	1.293204	I
1	-.81875	2456106.71682	1.290126	I
1	-.82175	2456106.71952	1.278397	I
1	-.81975	2456106.72023	1.275386	I
1	-.82475	2456106.72399	1.259742	I
2	1.54075	2456106.71613	1.293204	I
2	1.51525	2456106.71682	1.290126	I
2	1.56725	2456106.71952	1.278397	I
2	1.51025	2456106.72023	1.275386	I
2	1.54325	2456106.72399	1.259742	I
3	-.06125	2456106.71613	1.293204	I
3	-.04275	2456106.71682	1.290126	I
3	-.04775	2456106.71952	1.278397	I
3	-.06175	2456106.72023	1.275386	I
3	-.05075	2456106.72399	1.259742	I
4	-.65825	2456106.71613	1.293204	I
4	-.65375	2456106.71682	1.290126	I
4	-.64975	2456106.71952	1.278397	I
4	-.65375	2456106.72023	1.275386	I
4	-.64775	2456106.72399	1.259742	I

Table 3.2 The organized data from Table 3.1. The average magnitude of each box in Table 3.1 has been subtracted from every star within that box and re-organized according to coordinate number. The boxes shown here delineate different stars.

```
for(i=1;i<=num;i++){
       temp3 = 0.0;
       ss1 = 0.0;
       for(j=1;j<=tm1;j++){
          temp3 = temp3 + store[i][j];
          ss1 = ss1 + pow(10,(-0.4*store[i][j]));
       }
       for(j=1;j<=num2;j++){
          tm4 = remove[j];
          temp3 = temp3 - store[i][tm4];
          ss1 = ss1 - pow(10,(-0.4*store[i][tm4]));
       }
       ensem[i] = temp3/(tm1-num2);
       super[i] = (-2.5)*log10(ss1);
    }
    for(i=1;i<=num;i++){
       for(j=1;j<=tm1;j++){
          result[i][j] = store[i][j] - ensem[i];
          supres[i][j] = store[i][j] - super[i];
       }
    }
```

Figure 3.4 An excerpt from the code in varstar. This enables varstar to extract, average, and organize the data from each star.

and HJDs for one star in the same place.

To facilitate this process, brightER asks the user if they would like to run the script varstar. Varstar is a C script that outputs a file for each star containing its magnitudes and HJDs. It was modified from the original to include the Julian date represented by either 2450000 or 2440000. Varstar automatically calculates the average and super magnitudes for each frame and subtracts them from every star within that frame. In addition to this, varstar calculates the error per observation and asks the user if they would like to remove any stars from the ensemble. The ensemble refers to the stars used to calculate the average or super magnitudes.

Varstar outputs two main types of files with the magnitude and HJD. The first is a file with a DAT file extension. This file contains magnitudes that were subtracted by the average magnitude of every star in their frame. The second is a file with a SUP file extension. This file contains magnitudes that were subtracted by the super magnitude, obtained from the sum of all of the fluxes in their frame. The code for this process is shown in Figure 3.4. The matrix called *store* contains the value of each magnitude. The variable *j* refers to the coordinate number and the variable *i* refers to the frame number. Likewise, the variable *tm1* is equal to the total number of coordinates, and the variable *num* is equal to the total number of frames. Line 133 shows a for loop that runs for every star in a particular frame. Line 134 adds together each magnitude in the frame and line 135 adds together each flux in the frame. To find the flux of each star, varstar uses

$$F_j = 10^{-0.4*m_j} \tag{3.1}$$

where m_j is the magnitude of one star. Lines 137-141 remove any stars from these sums that were previously defined by the user. This process then restarts for the next frame until it reaches the end of the for loop in line 130. Line 142 shows a vector called *ensem* that stores the average magnitude for each frame. It divides the summed magnitude stored in *temp3* by the total number of stars used in the sum. Similarly, line 143 shows a vector called *super* that stores the super magnitude for each frame. The super magnitude is calculated by

$$m = -2.5 * log(F) \tag{3.2}$$

where F is the sum of all the fluxes in one frame. The for loop in line 146 runs through each magnitude in a frame and subtracts either the average magnitude or the super magnitude. It stores these in the matrices *result* and *supres*, respectively (see lines 147-148). Again, it does this for every frame. Finally, it uses a for loop to call each magnitude with the same coordinate number out of *result* or *supres* and outputs it to either a DAT or SUP file, along with the corresponding HJD.

Varstar displays a list of values on the terminal window (see Figure 3.5) These values are listed as *id, mag1, error1, mag2*, and *error2*. The *id* value is the equivalent of the coordinate number. *Mag1* and *mag2* show the magnitude values for a star. *Mag1* came from the average magnitude comparison, whereas *mag2* was obtained using the super magnitude method. The magnitudes shown in the figure are the average magnitudes for a star across all frames. The errors are computed using the standard deviation equation:

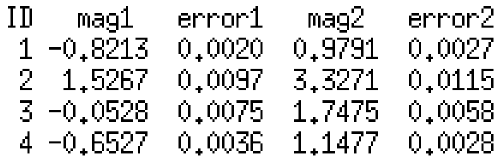
$$\sigma = \sqrt{\frac{\sum (x-\bar{x})^2}{n-1}} \tag{3.3}$$

where x is equal to the magnitude of one star, $\bar{x}$ is equal to the average magnitude of that star, and n is equal to the total number of magnitudes used to compute the average. The user may then remove any stars with a high error from the ensemble.

Varstar numbers the output files according to the coordinate number. If the user cannot remember which coordinate number refers to the target object, he or she can look in "ds9.reg" (the coordinate file) and see which line contains the coordinates for that object. At present, the version of varstar in brightER suppresses any output other than SUP files. However, this can easily be adjusted by uncommenting if the user wishes to use DAT files or any others. Unlike the IRAF scripts, varstar must be recompiled after editing the code. This is accomplished by typing the following into the terminal window: `gcc varstar5.c -o varstar -lm`. Varstar is the final script called by brightER. After completing this script, the entire brightER program ends and the user is free to plot the results.

3.3 Plotting

The user can plot the results from varstar using any method; however, the brightER package does include a script specific for plotting called plotsup. This is a MATLAB script that uses the SUP files to plot the magnitude vs. the HJD.



```
ID   mag1   error1   mag2   error2
 1 -0.8213  0.0020  0.9791  0.0027
 2  1.5267  0.0097  3.3271  0.0115
 3 -0.0528  0.0075  1.7475  0.0058
 4 -0.6527  0.0036  1.1477  0.0028
```

Figure 3.5 A table of the magnitude and error output from varstar. The first magnitude was computed using the average comparison, while the second was computed using the super magnitude comparison.

Plotsup labels the y-axis "magnitude," the x-axis "HJD," and names the title as the root name of the SUP file. The HJD is a very long number that only changes marginally from frame to frame. For this reason, it can be difficult to distinguish any difference in the HJD along the x-axis. Plotsup remedies this problem by removing any commonly occurring values in the HJD from the x-axis tick marks and instead displays them under the x-axis label.

An option appears at the beginning of the script to include a line of best fit. This is displayed on the plot as a red line. Plotsup uses a third-order Gaussian fit using MATLAB's built-in `fit` command. A comment in the code suggests using a fourth-order polynomial if the Gaussian does not do well. Additionally, plotsup shows a yellow line. This is a smoothing function connecting the data points together, which are displayed as black dots. Both the best fit and smoothing function are simply to give a better idea of the shape of the curve (see Figure 3.6).

Plotsup shows a plot for every file created by varstar. This means the user will have the same number of plots as stars originally selected. If the user wishes to view a specific plot, plotsup has the option to start from any file. This is especially useful when the user knows which file contains the target object. Otherwise, one can start from the first file and click through each, making it very easy to find new objects of interest. This is the final step in the photometry process. The user is now able to use these plots to glean further information about the observed objects.

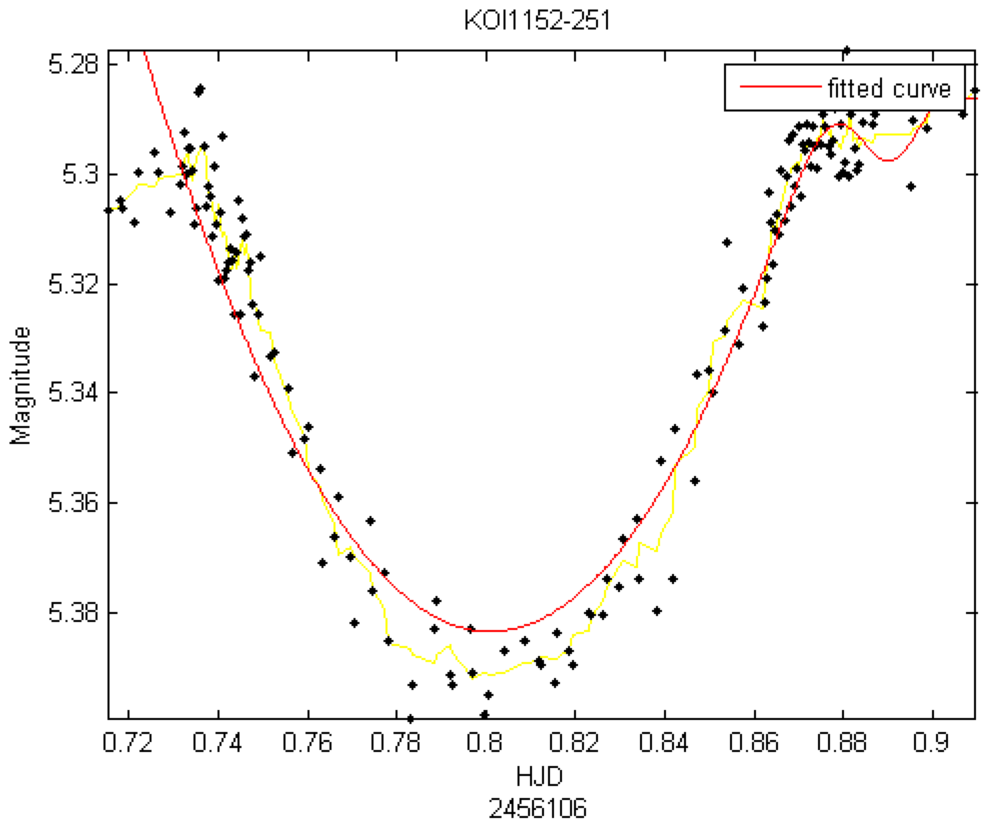


Figure 3.6 Plot of KOI-1152 in the V filter obtained from plotsup. The HJD is the sum of the value on the x-axis and the number listed under the “HJD” label. For example, the first point on this plot has an HJD of 2456106.71. The yellow line is a smoothing function, and the red line is a Gaussian fit. The number of black dots is equal to the number of frames.

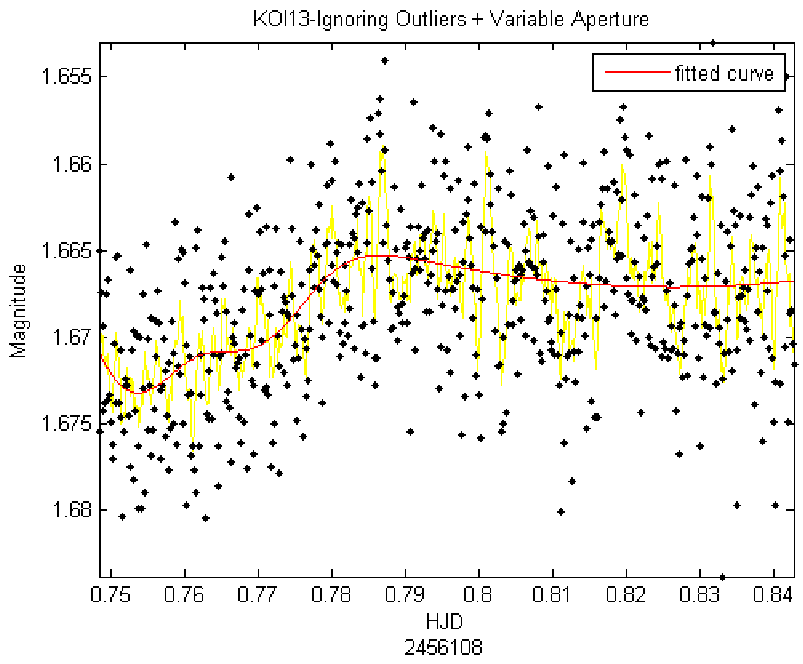
Chapter 4

Results and Conclusions

4.1 Results

The most important capability of brightER is its ability to produce light curves. We noticed that small changes in the aperture parameter drastically affected the shape of these curves (see Figures 4.1 and 4.2). It was for this reason we created four different versions of nphoter. It is yet to be determined whether a constant aperture or an aperture that varies based on the average FWHM of each frame is more accurate. Additionally, we seemed to get worse results when we removed outlying FWHM values from this average. Currently, brightER uses a variable aperture and does not remove outliers.

The greatest advantage of using brightER is the increased speed at which one can do photometry. Regfile was able to mark 408 stars in 45 seconds. In the same amount of time, we were able to mark only 7 stars by hand. Thus, marking 408 stars manually would take approximately 44 minutes. In just 30 minutes, brightER was able to create a coordinate file, phot 596 frames, and produce 176 light curves. This is equal to a photing rate of about 20 frames per minute, which is much quicker than the 3 frames per minute the average person can do. It would take just over 3



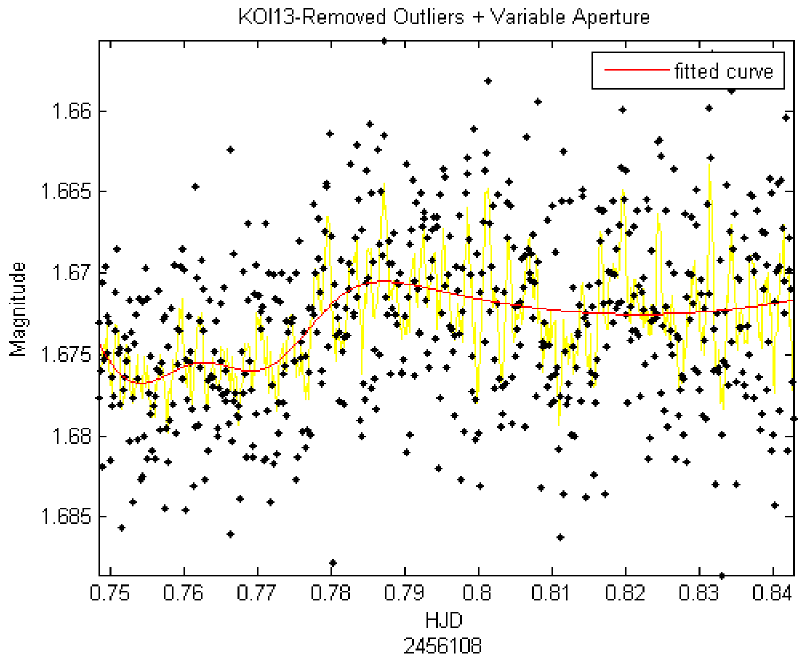
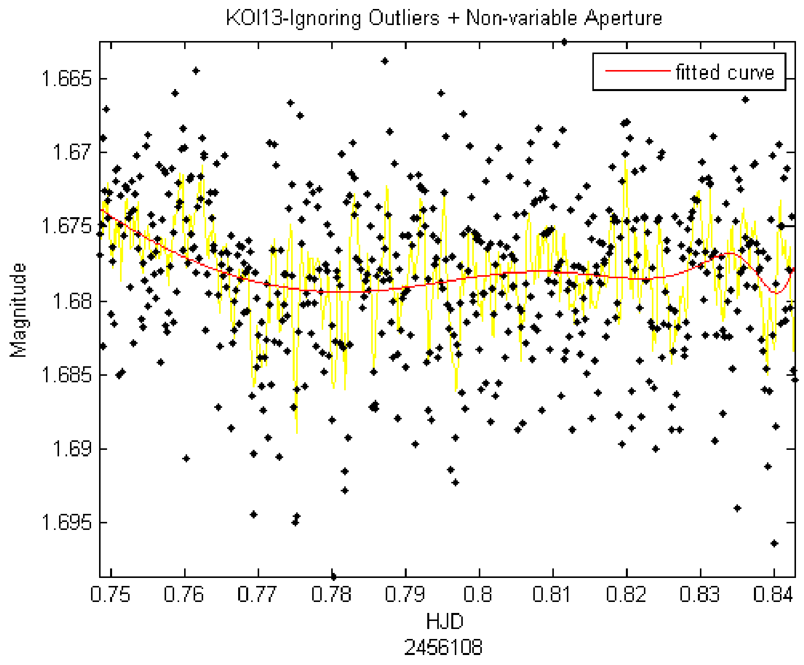


Figure 4.1 A comparison of the light curves of KOI-13 produced by two different versions of nphoter. Each of these were photed using a variable aperture. This shows a less distinct curve with the removal of outliers. Compare to Figure 4.2.



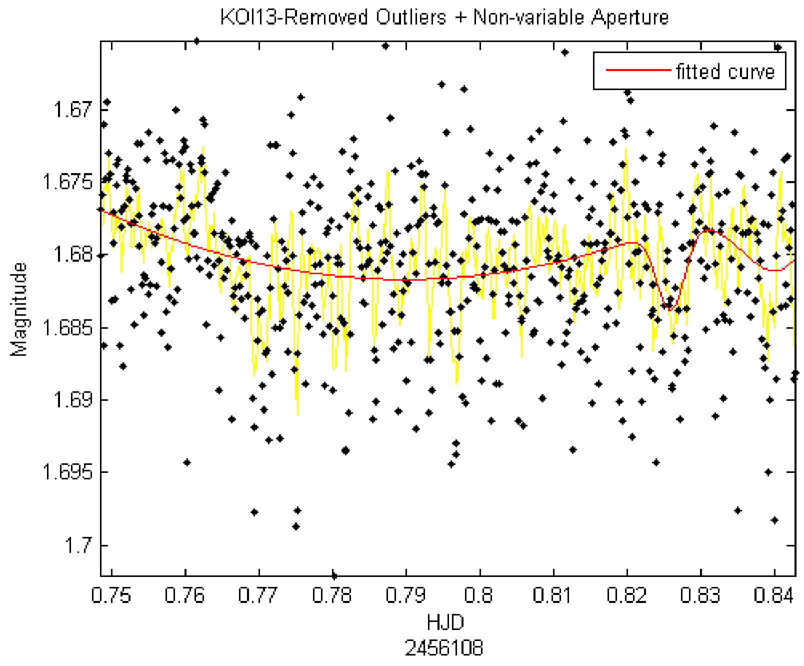


Figure 4.2 Two additional light curves of KOI-13 produced with the remaining versions of nphoter. Each of these were photed using a fixed aperture. Again, there is more scatter with the removal of outliers. The small dip shown in Figure 4.1 is not present in these plots, although the only difference during photing was the aperture parameter.

hours for the user to phot the same set of data by hand, without consideration for the amount of time to mark stars, organize data, and plot. From these results, we see that brightER saves the user an exceptional amount of time.

4.2 Future Work

Although brightER has significantly increased the ease and efficiency of the photometric process, improvements can still be made. We hope to find a way to better track the stars across the frames, or to align each frame so the stars do not shift at all. BrightER's error finding algorithm is a great start, but the goal is to eliminate errors altogether.

Other future work with brightER includes the analysis of the plots. We plan on adding error bars to help determine the accuracy of the results. In addition to this, we are working towards using a different method for fitting the curve, as the current fit does not give any information about the object.

Originally, brightER was created for the purpose of determining whether Kepler Objects of Interest (KOI) were extra-solar planets, but it can be used by anyone doing photometry. Currently, our research group is using brightER to follow up KELT transiting planet candidates with the David Derrick 16" Telescope at the Orson Pratt Observatory.

4.3 Conclusion

We created a program called brightER that uses several scripts to automate much of the photometric process. These are written mainly in IRAF's command language; however some are written in C and MATLAB. The tasks they perform include creating a coordinate file, displaying multiple frames, photing each frame while searching for errors, and organizing and plotting data. BrightER has improved these procedures by making them noticeably faster and decreasing the dependence on the user. We plan to improve the program by perfecting the tracking methods, adding error bars, and fitting the light curves with the appropriate equations. Our research group is currently using it to follow up transiting planet candidates, but other students are using it for galaxy and variable star research. BrightER enables astronomers to achieve their most desired goal: discovery.

Appendix A

Instructions for Running BrightER

1) Write the following lines into the login file (called “login.cl”):

```
task $sed=$foreign
task nphoter=‘‘File Path’’/nphoter.cl
task brightER=‘‘File Path’’/brightER.cl
task daofindsort=‘‘File Path’’/daofindsort.cl
task $varstar=$‘‘File Path’’/varstar
task viewframes=‘‘File Path’’/viewframes.cl
task regfile=‘‘File Path’’/regfile.cl
task rfprocess=‘‘File Path’’/rfprocess.cl
task fit2fits=‘‘File Path’’/fit2fits.cl

images
plot
dataio
lists
noao
digiphot
apphot
imred
ccdred
obsutil
```

2) Replace “File Path” with the location of the files.

Example: `task nphoter=/data/ecrouch/Research/Scripts/brightERpkg/nphoter.cl`

3) Change the directory to that containing “varstar5.c.” Compile varstar with the following code:

```
gcc varstar5.c -o varstar -lm
```

4) Edit the following parameters in IRAF under `noao/apphot/datapars` **with the following code:**

```
epar datapars

exposure = “Keyword relating to exposure time”    (i.e.  exptime)
airmass =  “Keyword relating to airmass”          (i.e.  airmass)
filter =   “Keyword relating to filter”           (i.e.  subset)
obstime =  “Keyword relating to observation time” (i.e.  hjd)

:wq
```

Troubleshooting and Further Information:

1. BrightER changes the parameters LTV1 and LTV2 in the image header by using hedit. The value expression changes to 0. These parameters are just offsets that mess up the centering file. BrightER also changes the maxshif parameter in centerpars to 3, meaning if the coordinates have shifted more than 3 pixels, you’ll most likely get errors when photing. Typically when it has shifted by that amount, the coordinates are no longer on the star and an error is necessary.

2. If errors for the same coordinates appear multiple times, there are most likely coordinates on oversaturated stars, stars with two coordinates, or coordinates on cosmic rays. This becomes problematic for `phot` and `psfmeasure`. Remove these bad coordinates and restart nphoter. To avoid this problem in the future, carefully scan the the coordinates before saving the coordinate file.

3. When DELETING a coordinate after running nphoter: completely restart nphoter, deleting

any magnitude files the program may have created as well as the created file "originalcoord.reg". The reason for this is that the number of coordinates in the magnitude files will not be equal, resulting in errors while running varstar.

4. When reloading “ds9.reg” to shift coordinates in nphoter, make sure to remove any coordinates already on the frame. Otherwise, both sets of coordinates will be saved. It might be easier to shift the coordinates already on the frame and re-save them.

5. Regfile uses IRAF’s built-in function `imexamine`. When stopping regfile in the middle of running, `imexamine` processes will still be running. Hit "q" on the DS9 frame to exit out of `imexamine`. The next time regfile is run, it gives an error: ‘‘`ERROR: floating divide by zero.`’’ Logout of IRAF and log back in. See `help imexamine` in IRAF for more information.

6. If the error ‘‘`Cannot open device (node!  Imtool,,4096,4096)`’’ appears, it means that DS9 is not open or IRAF cannot access it. Simply reopen DS9 and restart brightER.

7. IRAF needs the permissions to be able to write and read files. To check file permissions, type `ls -l` into the linux terminal.

8. To force quit, hit ctrl-c.

Bibliography

Foster, G. 2010, Analyzing Light Curves: A Practical Guide (Raleigh, NC: LuLu)

Howell, S. B. 2011, Astrophysics and Space Science Library, 373, 71

McLean, I. S. 1997, Electronic Imaging in Astronomy: Detectors and Instrumentation (West Sussex, UK: Wiley)

NOAO. Accessed June 17, 2013, http://iraf.noao.edu

Ryden, B., & Peterson, B. 2009, Foundations of Astrophysics (New York, NY: Pearson Addison-Wesley)

SAO. Accessed June 17, 2013, http://hea-www.harvard.edu/RD/ds9/site/Home.html

Sterken, C., & Manfroid, J. 1992, Astronomical Photometry, A Guide (Dordrecht, Netherlands: Kluwer Academic Publishers)

Index