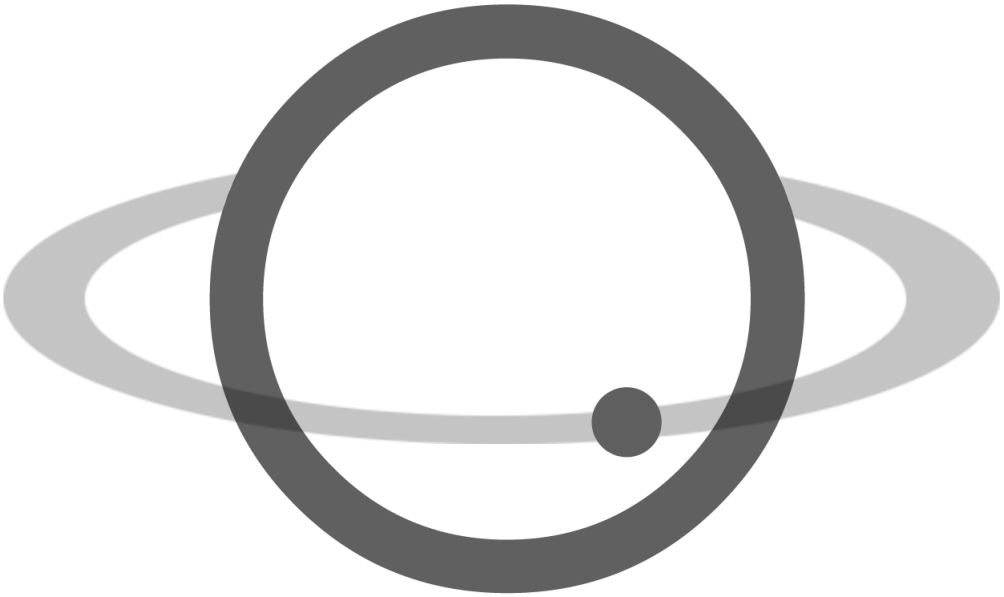
OSCAAR: Open Source Differential Photometry Code for Amateur Astronomical Research

Brett M. Morris and the OSCAAR Team

May 27, 2013



Contents

1 Introduction

OSCAAR 's core is a differential photometry package built for users of any experience level at observatories of any size. The process of differential photometry can be applied to many astrophysical phenomena including transiting extrasolar planets, variable stars, rotating asteroids and more. This package is prepared specially for transiting exoplanet observations, though often other photometric observations can be analyzed with OSCAAR without any tweaking to the source code.

Development for OSCAAR began in the summer of 2011 to take a series of images obtained at the University of Maryland Observatory in College Park, MD, USA and churn out light curves of transiting extrasolar planets. If you're looking to do something similar, you've found the right code! The core of the code is written in Python, but it has been designed to be operated with a graphical user interface (GUI) for users who have never seen Python before. That being said, experienced programmers will find that the guts of OSCAAR provide a well-documented toolkit for even the most demanding of photometric measurements. Since the v1.0 release, OSCAAR has been used by astronomers at all levels, from undergraduates to observatory-directing IAU members.

Here we summarize the contents of this documentation. If you are new to photometry, you may consider reading Section 2, which will summarize observing techniques to make successful photometric measurements. In Section 3, we will detail how to run OSCAAR from the graphical user interface (GUI) or other ways without directly coding in Python. In Section *** we discuss the classes and methods that are built into OSCAAR that users with programming experience may find useful to design their own analysis tools.

1.1 System Requirements

There are several packages in addition to Python that must be installed on your machine before running OSCAAR . Most of these packages have been around for some time in the astronomical community. They are all free and available for download via links below. We apologize for the number of packages that are necessary to download, but we have decided that these packages enable for the most efficient open-source distro that we can provide.

Python 2.7: The core language of oscaar
NumPy 1.6.0-py2.7: A Python module for efficient scientific computations
PyFITS 2.4.0: A Python module for handling FITS files
Matplotlib 1.X-py2.7: A Python module for plotting
wxPython: A Pythonic GUI toolkit
PyEphem (*optional*): A Python module for ephemeris generation

PyEphem is not required to run differential photometric analysis with OSCAAR but enables you to make detailed plans for observing nights in advance. More on this in Section ***.

1.1.1 Disclaimer

Though the modules used in this package have been stable historically and probably will not go away any time soon, we want to note here that changes implemented in newer versions of these packages could potentially break features in OSCAAR . If you notice a broken feature, let us know by posting an issue on our Issue Tracker (see Section 5)

2 Collecting Data

2.1 Picking A Target

2.2 Navigating The Sky

It is one hour before the photometric event that you want to observe, you have taken your flat fields, and you're ready to slew to your target. You punch in the RA and declination of the target, and your telescope lumbers over to that part of the sky, but you can't recognize the pattern of stars that come out on your first image. Is that star near the center of the field actually your target, or are you looking at the wrong part of the sky?

While it is not necessary for running `OSCAAR` , we highly recommend that you download the open-source planetarium software Stellarium for navigating the sky and planning. This free, user-friendly package is supported on nearly every operating system and boasts some advanced features that will make it easy for you to find your targets.

One feature of particular utility for the exoplanet community is that the object finding search bar that you use to find objects in Stellarium is linked to the professional astronomical database SIMBAD to resolve nearly any object by any name. SIMBAD is updated relatively quickly, so when you want to go out and observe that new exoplanet but your old planetarium software doesn't know about the latest WASP or HAT target, Stellarium will resolve the name through SIMBAD and take you to the coordinates published for that object[1].

Stellarium also has great "Flip Scene" buttons that allow you to mirror flip your views of the sky to match the flips that your telescope optics do. That way you can match up your observed field of view with the stars in Stellarium without having to imagine complicated image transpositions in your head.

2.3 Telescope Tracking

Differential photometry is generally done on a series of images of the same patch of the sky over a long period of time. You'll need a telescope that is well polar-aligned so that the telescope's tracking keeps the stars in nearly the same spot on your detector throughout the duration of your observations. It is a tremendous challenge to align most small telescopes well enough to track a star perfectly over several hours, so `OSCAAR` is built to monitor and correct for the drift in star positions on the detector over time. Just make sure your target object and a few comparison stars stay in the field throughout the whole observation[2]. The star tracking algorithm follows each star individually over time. If your photometric target is an asteroid, for example, that moves relative to the comparison stars over time, `OSCAAR` will happily track the asteroid and the comparison stars independently.

2.4 Defocusing

In precision photometry, relying on individual pixels is dangerous. Pixel defects occur frequently that can cause a pixel to read much higher counts than they actually receive (these are called "hot pixels") or sometimes much lower counts ("dead pixels"). If your target object is perfectly focused on a few pixels, you may be putting all of your photometric-eggs in one basket. Thus it is often advised that you **do not focus the telescope perfectly**

[1] Of course, this feature only works when your machine is connected to the internet, so if you do not have an internet connection at your observatory, you'll still need to query for your target before you get to the observatory, print out your star charts.

[2] If you need to re-center your target in the middle of an observation, `OSCAAR` will only be able to continue to track the stars without special code tweaking if you change the stars' centroid positions by a few pixels (i.e., ∼2 or 3 pixels) at a time in between each exposure. **The current version of `OSCAAR` is not resilient against sudden discontinuous tracking anomalies.**

when doing photometric measurements. If you can, defocus the telescope significantly so that you spread out the starlight over a few of pixels, and your pixel-to-pixel variations will play a less-significant role in the introduction of unwanted systematic noise and bias. See Figure 1 for examples.

Some photometry codes prefer objects focused in Gaussian-like shapes, but `OSCAAR` is written to accurately track stars of unconventional shapes. At the University of Maryland Observatory, we make most of our measurements on a 6in refractor. We defocus the stars until they look like small donuts (the hole is an artifact of the optical path of the refractor), and we've found that our photometry comes out best this way. `OSCAAR` won't complain if your stars are donut-shaped, Gaussian or somewhere in between.

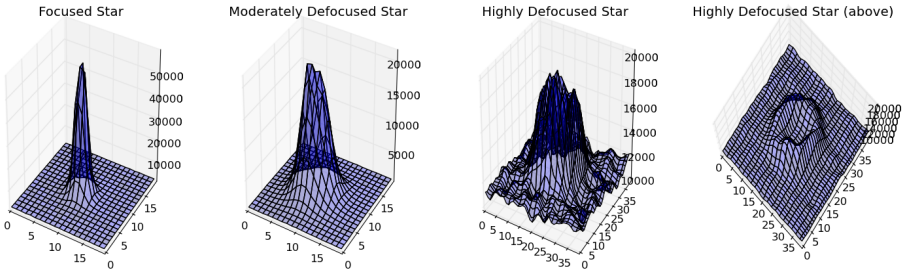


Figure 1: Images of stars with varying levels of defocusing. The x and y axes are the pixel indices, intensity in counts is plotted along the z. `OSCAAR` can track any of these stars with precision – even when the shape of the star is not approximately a Gaussian, like the highly defocused star on the right which is shaped like a donut.

One must keep in mind that by spreading out the light over many pixels, the counts measured by each pixel is lower. As a result, defocusing is most helpful when observing bright objects. If the source you are observing is bright enough, you can defocus significantly without losing too much signal. However, it is important to note that the quality of your light curve will be directly related to how much more signal than noise you detected, as we will discuss in Section 2.5. Use defocusing sparingly (or not at all) when imaging dim targets[3].

2.5 Theory: Photon Noise

There is a fundamental physical limit to the signal-to-noise ratio that can be achieved in photometric measurements. Since photometry is the process of counting photons, there is a type of statistical uncertainty introduced into each measurement that goes by many names: *photon noise, Poisson uncertainty, shot noise,* etc. This uncertainty is unavoidable in counting measurements and is easy to calculate – the photon noise, or the uncertainty in a measurement of N photons, σ_N is

$$\sigma_N = \sqrt{N}. \tag{1}$$

You can see directly from Equation 1 that the fractional uncertainty in the signal (σ_N/N) decreases with increasing N as $1/\sqrt{N}$, so the more signal you have, the lower the limiting fundamental uncertainty. Of course, in practice there are many additional noise sources that will add uncertainty into your data and increase the scatter in your light curves.

[3]For example, we use a sharp focus for stars dimmer than $V = 10$ on our 152mm refractor at the University of Maryland Observatory.

2.6 Dark Fields and Flat Frames

Collecting dark frames and flat fields is standard practice for removing systematic effects from CCD images. Flat fields correct for dust and other inconsistencies in the optical path artificially brighten or dim the objects that you image. Dark frames remove some flaws in the CCD like hot pixels and dark current variations. Some CCD imaging software like MaxIm DL have easy preset routines for taking dark frames. We recommend taking ≥ 8 dark frames for each set of observations you take. OSCAAR will take the mean of these sets and apply them appropriately to each image of your data set.

Collecting good flat fields can sometimes be more of an art than a science, but good flats are important for good photometry. For this reason, we've incorporated two methods of "master flat" making routines into OSCAAR , that combine raw flat fields into one master flat. The first method is called the "Standard" method by the OSCAAR GUI, which takes a mean of all of the input flat frames. This is the fastest computational method. Standard master flats are ideal for "dome flats", where the flat fields are obtained by imaging a screen. Many astronomers prefer "twilight" or "sky" flats, in which the telescope takes images of the sky as the sun sets, and light from the sky acts as the uniform light source. We developed a corresponding twilight master flat setting for OSCAAR , which fits a linear function to the intensity of each pixel over time, and uses the best-fit intercept as the normalizing factor for the master flat. Since loops are slow in Python, this method may take a few minutes to produce a master flat, but the payoff that you gain in photometric precision is worth the wait. At the time of writing, we're currently investigating how to implement these algorithms in Cython to help us speed up these expensive computations.

2.7 Picking A Target

2.7.1 Transit Predictions

So you're planning a night of transiting extrasolar planet observations, and you need to know what planets are transiting during your local night. We recommend using the Czech Astronomical Society Exoplanet Transit Database's Transit Prediction tool. If you enter your latitude and longitude, this web-tool will tell you which bright transiting extrasolar planets will be visible and transit on a given night from your location. This tool is invaluable for planning. It's also great because you can contribute to this database by submitting your light curves to help constrain the orbital parameters of the planets. With a telescope and OSCAAR , you can contribute to real science!

2.7.2 Choosing The Field

When you chose your target for differential photometry, you need to be sure there are other stars in the imaging field. Here we'll define some terms that are important from here on:

Target Star: The target of your observations – the host star to an exoplanet or the variable star that you're measuring for intrinsic variations in luminosity. Of course, this "star" could be an asteroid, if you're into that sort of thing.

Comparison Star: A star other than the target star that you will use as a basis for determining the intrinsic variations in the brightness of the target star. The comparison star should be one that is not known to have

intrinsic variations.

There must be at least one comparison star in order to do differential photometry, and the more the better. There is no magic number of comparison stars to have, but if you have the opportunity to fit more control stars in the field by rotating your CCD or effectively "zooming out," it will be worth your while. Based on prior experience with OSCAAR , more than two good stars will suffice, but 10 can give you great results. Later, we'll discuss how to know which ones are "good."

Photometry of bright stars is always preferable to dim stars, but of course there are more dim stars than bright stars in most of the images you will take. In order to avoid uncertainty introduced noise with very dim stars, pick control stars with peak intensities more than double the average background intensity in the area surrounding the star.

Different transiting exoplanets change the brightnesses of their host stars by different amounts. This parameter is called "**depth**" and is often measured in units of millimagnitudes (mmag). The greater the depth of a transit, the more likely you will be capable to detect the transit. Since some of these depths are so small, you might try measuring short period pulsating (SPP) variable stars with high amplitude luminosity oscillations (like YZ Boo) before you move on to exoplanets. Once you characterize your ability to measure the large intrinsic variations of variable stars, you will be able to characterize your ability to detect the small depth regime of transiting exoplanets.

2.8 Imaging Software

A bunch of imaging packages could suite your needs for photometry with OSCAAR . At the University of Maryland Observatory, we use MaxIm DL to handle our imaging. Any imaging software that enables you to take a time-series of CCD images will do.

Observing software like MaxIm DL allow you to choose your imaging **binning**, which enables the detector to read-out pixels in groups. For example, a 2×2 binning will take a square of four pixels and save them as one composite pixel. This reduces the read-out time, the size of the output files, and the run time of scripts that have to read and manipulate those images. We recommend that you use 2×2 binning when you can for these reasons. It is often unimportant to save images at the full-resolution of the detector, especially when you are using defocusing anyway, as described in Section 2.4.

3 Running OSCAAR v2.0

The first step in running ocaar 2.0 is to execute the oscaarGui.py python file which is located in the OSCAAR folder that you downloaded. This should bring you to a screen that looks something like this:

****Insert Screencap here****

You will now be shown some initial parameters that you must set in order to run OSCAAR .

3.1 Locating Your Stars with DS9

The first and most intensive task you'll have to do to prepare OSCAAR for analysis is to tell it what stars you care about in your images. If you have a set of images, there could be tens or hundreds of stars in each image, some of them close to the edges, some of them with binary companions; some of them ideal control stars, some of

them not. In order to ensure that the stars being picked as control stars are appropriate choices, OSCAAR has the user enter each of the stars into OSCAAR with the help of SAO Image DS9 (see Section 1.1 to download).

To begin, start DS9 and open the first image from the series of images you will process. You can also start DS9 from within OSCAAR by clicking Open DS9 button. Set the scale and zoom so that you can see most of the image and most of the stars in it. With the mouse set to Pointer (Edit >Pointer), click on the target star (the exoplanet host star or variable star). A green circle will appear over the star along with a dialogue box which contains the pixel coordinates of the circle's center and radius, see Figure 2.

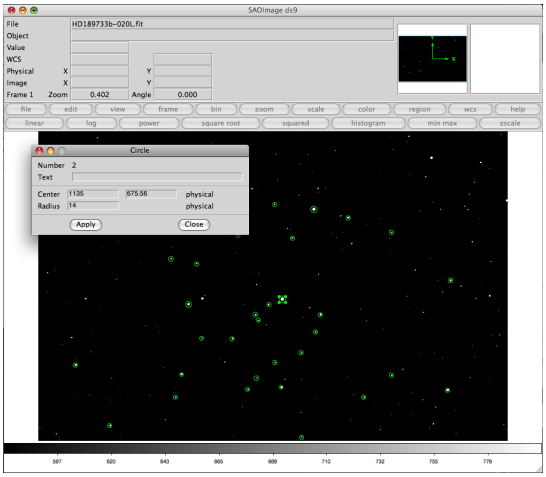


Figure 2: Using DS9 to locate the target and comparison stars.

The first star that you select will be treated as the target star, and any subsequent choices will be comparison stars. Repeat this process for as many comparison stars as you like (try to keep it under 25 for speed). Avoid picking stars near any obvious defects on the detector, any stars less than 150 pixels from the edge of the image, or stars with a neighboring star close nearby.

When you're done choosing control stars, go to Regions >Save Regions... and save a regions file in a directory where you will be able to find it later. This file will contain the pixel coordinates of the stars at the beginning of the observation, which tells OSCAAR which stars to track. You may want to check that the test star is in fact the first star in the regions file. Open the regions file in a text editor and check that the first line resembling "circle(100,600,10)" has the proper (x,y) pixel coordinates and radius (in this example, the position is (100,600) with a radius of 10). If the first circle coordinate line does not point to the target star, find the circle coordinate that does, move that line to the top of the list, and save the file.

In the OSCAAR GUI, click Browse in the Path to Regions File field and enter the path to the regions file you just made.

3.1.1 Locating Your Data

OSCAAR is built to find files by their paths. To properly set the image paths, you should click the browse button next to the dark images path field and choose all of the dark image files that you wish to be processed. To do this, select your first file, then hold the Ctrl key on Windows or the Command key on Mac and click every file you would like to select. Alternatively, if they are all listed consecutively, you can click the first file, hold shift,

and then click the last one. Now click the "Open" button and all of your dark images should be listed separated by commas. Do the same for your data images in the `Data Images` field of the `OSCAAR` GUI.

3.2 Making a Master Flat

Now that you have properly set the path to the darks and the data images, you must now make your master flat. First click the "Master Flat Maker" button at the bottom of the `OSCAAR` window, and a dialog box should pop up that looks like Figure 3.

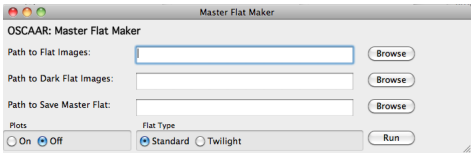


Figure 3: `OSCAAR` 's Master flat maker allows you to make master flats out of raw images.

Now you must set the paths to the flat images and the dark flat images in the same way you set the image paths previously. You must also enter the path and filename for the master flat. To do this, click the `Browse` button and navigate to the location you would like to save the master flat.

There are two options for how you can make your flat, either the "Standard" or "Twilight" flat type. The standard flat maker algorithm is a simple mean combination of the flat frames that you enter into the flat maker GUI. If you took dome flats or bright sky flats, this is the right option for you. If you took flats at twilight as the sun was setting and the sky background was your "screen", the twilight flat algorithm will fit a linear function to the intensity of each pixel over time, and use the intercept as the normalization factor for the flat. The twilight flat algorithm is much slower than the standard method, because those linear fits are computationally expensive in an interpreted language like Python. We hope to code up alternative versions of these routines that you can experiment with on your dataset, some of which may access much faster, compiled code in C or Cython.

Now click the `Run` button on the master flat maker and close it when it is done. You should now choose the output file when selecting the path to the master flat frame in the main `OSCAAR` GUI.

3.3 Running Parameters

The next step in running `OSCAAR` is to properly set all of the initial running parameters. These have default values set, but these may need to be modified for your particular data set.

3.3.1 Smoothing Constant

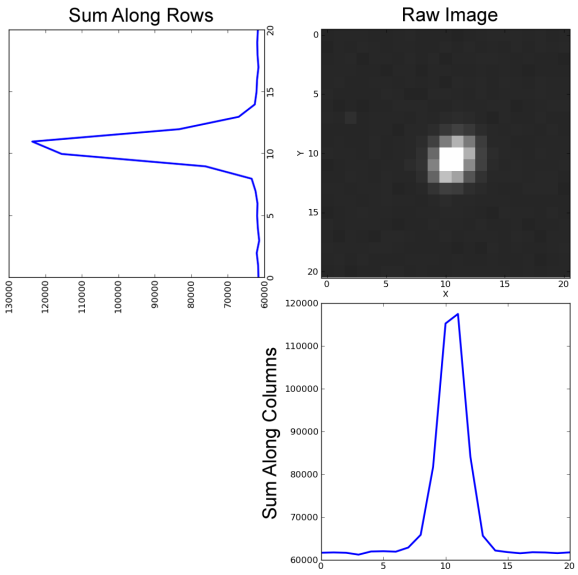


Figure 4: Sums along the columns and rows around a star.

By default, oscaar v2.0 tracks stars with a fun algorithm described in Section 4.2. This algorithm finds the centroid of the star by looking for certain features in the first numerical derivative of the spatial intensity profile of the image of the star. These "intensity profiles" are the sums of pixel counts along the rows and columns near the star being tracked, see Figure 4 for an example. In order to find the centroids accurately using this method, it helps to smooth out the stars by blurring the image artificially, so that background noise and bad pixels are not incorrectly interpreted as features of the stellar intensity profile. Of course, the artificial smoothing is non-destructive; the images used to measure photometry are not smoothed.

Since it is not uncommon to defocus telescopes for photometry, you may want more or less smoothing in your images. The running parameter `Smoothing Constant` adjusts how much smoothing is applied in the tracking algorithm. `Smoothing Constant = 0` will not smooth the image at all (not recommended), and values around ~ 3 will be significantly smeared (non-integer values are accepted). See Figure 5 to see what how various values of the `Smoothing Constant` affect a raw stellar intensity profile.

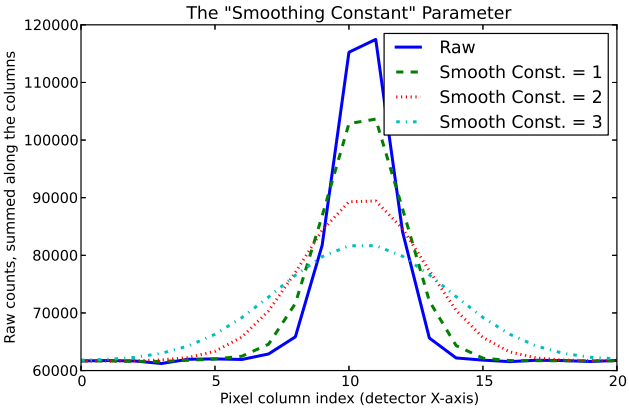


Figure 5: A range of `Smoothing Constant` values from 0 to 3 are applied to the stellar intensity profile along the columns surrounding one star (see Figure 4 for the origin of the intensity profile).

3.3.2 Tracking and Photometry Plots

In order to visualize the effects of the running parameters that you entered on the photometric process, you can have oscaar plot visualizations of various procedures as they are executed. This will add significantly to the runtime of your analysis, as the plotting package `matplotlib` is not as fast as `OSCAAR` 's algorithms, however it will enable you to hone in on which running parameters you need, and allow you to troubleshoot if the results produced by oscaar are not what you expected. There are two plot settings to choose from: `Track plots` and `Photometry plots`. `Track plots` will show you the centroid solutions as they are calculated in real time, along with some guide lines, see Figure 6 for an example.

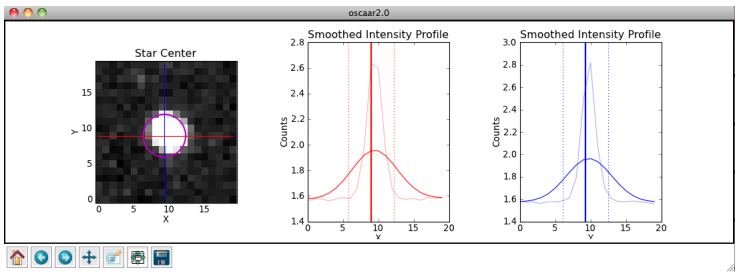


Figure 6: `Track plots` set to `True` and `Photometry plots` set to `False`. The leftmost image shows the star with a cross-hair indicating the centroid, also circled in magenta (the radius of this circle is not meaningful). The plots on the right indicate the sums of the intensities in pixels along the rows and columns, in the transparent curves. The solid curves represent the smoothed intensity profiles, which are used to find the centroid. The bold vertical lines mark the best-fit stellar centroid in each axis.

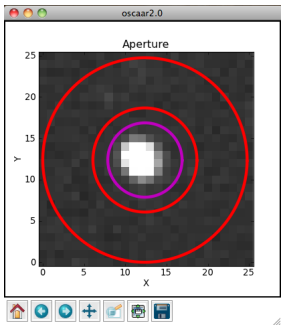


Figure 7: `Photometry plots` set `True` and `Track plots` set to `False`. The innermost circle (magenta) marks the `Aperture Radius` (see Section 3.3.5) measured from the stellar centroid found for this star by the tracking algorithm. All pixels that fall inside of this radius are summed to calculate the flux from this star. The next two outer concentric circles (red) circumscribe the "sky annulus," within which the sky background is measured.

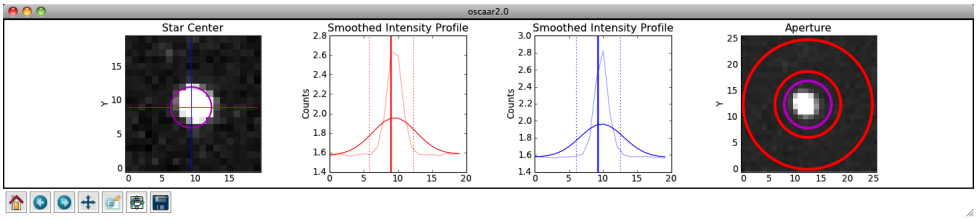


Figure 8: `Track plots` and `Photometry plots` both set `True`. See Figures 6 and 7 for more details on each subplot. Note: the `Smoothing Constant` may be too large on this image – see for example how broad the smoothed profile is compared to the raw flux.

3.3.3 Track Zoom

`OSCAAR` tracks each star through consecutive exposures by looking for the each centroid near to the detector location that it had in the previous exposure. Since the stars will drift somewhat from frame to frame if your telescope is not perfectly aligned, we define a width around the previous centroid within which to search for the centroid in the following frame. The ideal width of this search box will depend on the plate scale of your device, how defocused the telescope is, how poor the alignment is, etc.

See, for example, Figure 6. The `Track Zoom` parameter here was 20 pixels, as you can tell by the width of the image sampled along both axes. The limiting factor in how small the `Track Zoom` parameter should be is how far the star will move between this exposure and the next. The columns/rows where the slope of the intensity profile is at extrema (see the vertical dotted lines in Figure 6) need to fall inside of the image when cropped down to the size of `Track Zoom`. If the stellar centroid moved by 10 pixels in the next exposure for the star and zoom in Figure 6, `OSCAAR` would not be able to find one of the extrema required to find the stellar centroid, and the tracking algorithm would not produce meaningful centroid estimates.

3.3.4 CCD Gain

The gain of a CCD is a hardware parameter that is specific to your CCD and the possible settings that you have set on it. Without getting too much into detector physics, CCDs essentially convert photons to electrons, amplify the number of electrons, and then count the amplified electrons. The number of amplified electrons counted is often more briefly referred to as the number of "counts", or less transparently as "analog-to-digital units (ADU)". If we want to be precise about how we propagate uncertainties, we need to know how many electrons were on each pixel before amplification, so the detector gain comes into our calculations.

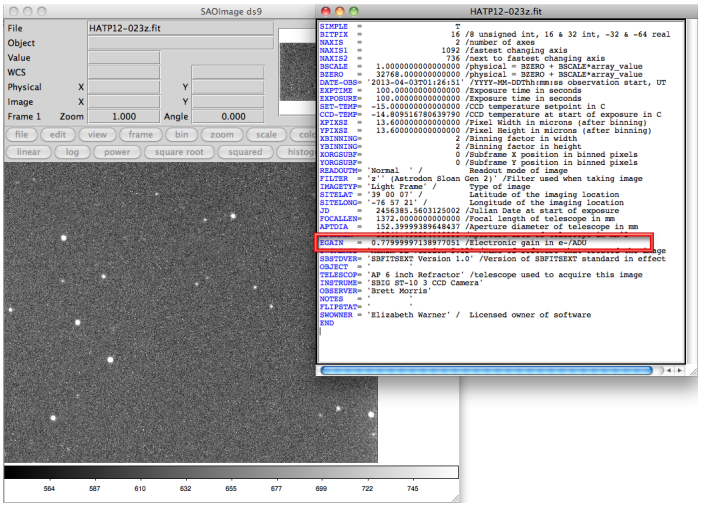


Figure 9: **Finding the gain**: Here's what the FITS header looks like when viewed in DS9, produced by Elizabeth Warner's 152mm refractor with an SBIG ST-10 CCD at the University of Maryland Observatory. The detector gain, labeled on this CCD as "`EGAIN`" is highlighted with a red box.

To find the gain of your detector, open one of your images in DS9. Click through the drop down menus from `File > Display Fits Header`. The Fits Header is a text file within each FITS file that records some notes on every exposure you take. Often there will be a line called `EGAIN` in there, with a comment resembling "/ Electronic gain in e-/ADU". This is the gain parameter to enter into oscaar.

3.3.5 Aperture Radius

The `Aperture Radius` is the radius measured outward from the stellar centroid within which we will measure the flux from the source. The `Aperture Radius` should fully enclose the source, and if possible, not extend any farther than necessary (though determining what is "necessary" may require some experimentation).

In the duration of an observation, changes in earth's atmosphere can cause the "width" of a star on your detector to be larger or smaller than it was at the beginning of the night. If the star gets wider over time, the dim outer edges of the star may begin to exceed the boundaries that you set by the `Aperture Radius`, and some of the flux will not be counted. The differential photometry script would interpret this as the star getting dimmer, which would ruin your light curve. Therefore, when selecting the `Aperture Radius` parameter, we suggest that you experiment with it. Try one value with the `Tracking Plots` turned on (see Section 3.3.2 for

details) and watch the apertures as they are applied to the images of the stars. Try a few different values (they need not be integers), and pick the smallest aperture that is sure to catch the whole star.

3.3.6 Notes

This text field is meant for you to enter any notes you might want to know later. All of your running parameters will be saved in the data file so you won't need to copy them here, but if you want to label the run in any particular way, there's a great little box here specifically for that purpose.

3.3.7 Ingress and Egress Times

If you are observing a transiting exoplanet, `OSCAAR` can do its most precise photometry if it knows when your target is "in-transit", meaning the planet is occulting the disk of star, or "out-of-transit", meaning the planet is not occulting the star. This is because it uses mathematical techniques that look for changes in each comparison star compared to the target star in order to determine which comparison stars are the best to use (see Section 4.3.2). If you compared the target to comparisons while the planet was in-transit, there would be a real (and important) difference between the two that we want to measure accurately. For this reason we input the ingress and egress times and only compare the target and comparison stars during out-of-transit exposures.

Enter the times in MM/DD/YYYY; hh:mm format, where the hours are on the 24-hour scale, in Universal Time. Seconds are insignificant.

4 Algorithm Notes

4.1 `oscaar.photometry.phot()`

4.2 `oscaar.astrometry.trackSmooth()`

The "**point spread function**" (PSF) for a particular observing setup is the shape of the image of a perfect point source (like a star, for example), which in practice is never a perfect "point" – it will always have some radial spread. For well-focused telescopes, the PSF usually resembles an Airy function, which can be well-approximated by a Gaussian. However, as suggested in Section 2.4, you may not always want to focus the telescope perfectly. Each telescope will produce a different PSF when significantly defocused, so assuming a Gaussian PSF would prevent us from using `OSCAAR` on defocused observations. This was a problem that we found with `OSCAAR` v1.0, so we developed a new method with some excellent advice from Professor Drake Deming.

`trackSmooth()` does centroid-finding by summing up the intensity of the pixels near the star along both the rows and columns. The sums along the rows and columns will produce intensity profiles in two axes similar to Figure 4. One feature of these profiles that holds for even strongly defocused PSFs is the sharp rise and decline in the intensity of the star on either side of the stellar centroid. Typically there is a well-defined absolute maximum and minimum in the first numerical derivative of the sum along the columns, as in Figure 10. The trick here is to take the midpoint between those extrema as the centroid, since searching for maxima and minima are computationally cheap.

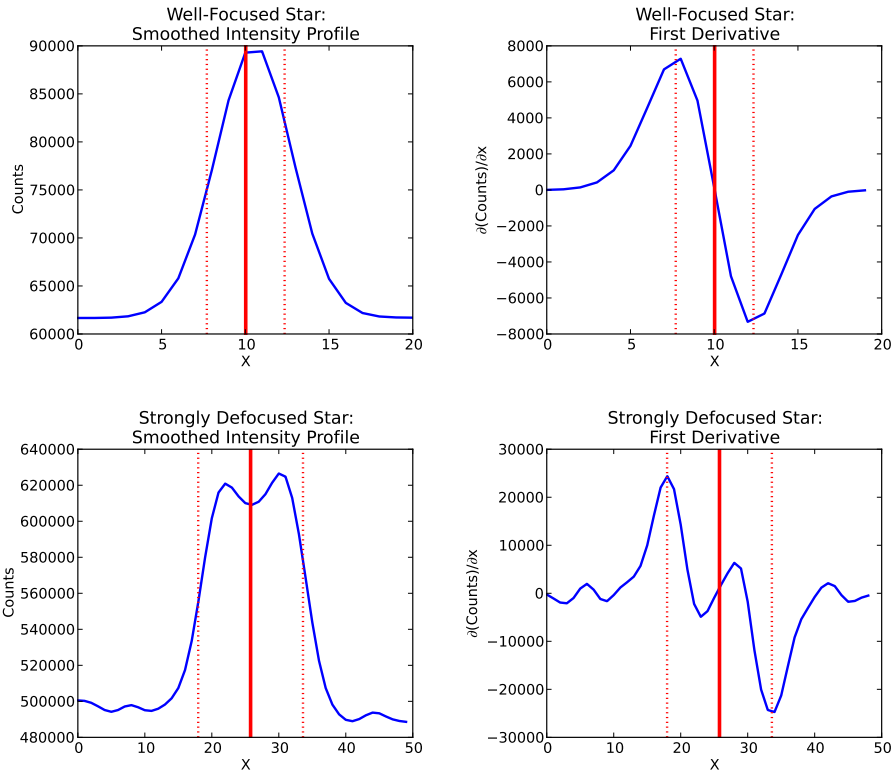


Figure 10: **Stellar Centroid Tracking**: The top row of plots show the sum of intensities in each pixel along the columns a well-focus star, similar to Figure 4, and the bottom row shows the same for a highly defocused star (see discussion on defocusing in Section 2.4). The plots on the left show the counts on the detector summed along the columns of pixels, and smoothed with non-zero smoothing constants. The plots on the right show the first spatial derivative of the intensity profiles, with clear extrema at the pixel locations corresponding to the points of greatest positive and negative slope in the intensity profiles. A quadratic is fit to the three points closest to each of the extrema, yielding sub-pixel precision on the coordinates of the extrema (marked above with the dotted vertical red lines). The midpoint between these best-fit extrema is taken as the stellar centroid, marked with the solid vertical red line in the plots above. This process is repeated for the other axis (sum along the rows) to get the other of the two dimensional centroid coordinates for each star.

In order to increase our centroid precision, we then use a little linear algebra to fit a quadratic to the three pixels nearest the maximum or minimum. The midpoints between the apexes of the best-fit quadratic near the maximum and minimum is taken as the centroid. This process is repeated for the sums along the rows and the columns to get the centroid in both axes.

Observational data is never clean. There may be dead pixels or hot pixels that would appear as sharp spikes or troughs in the intensity profiles. These pixels, if unaccounted for, will appear to the tracking algorithm to be the sharp edges of a star in the image. This is why we run the image through the smoothing routine – to smear out any sharp artifacts in the image, so that the tracking algorithm only anchors itself on the overall features of the image and not faulty individual pixels. This smoothing routine can apply varying degrees of smoothing to

each image, and choosing a proper amount of smoothing is covered in Section 3.3.1.

4.3 The `dataBank` Object

Storing and managing all of the data that comes with differential photometry scripts can be a nightmare, so we've developed the `dataBank` class to keep things organized. Python is an *object oriented* language, which means we can make *objects* in it. For programmers new to objects, we'll review them a little in the next section.

4.3.1 Quick Introduction to Object-Oriented Programming

One way to think about objects in terms of their use within `OSCAAR` is that they are a name given to a template for a collection data and functions that can be run on those data. For example, let's imagine an object called the `car` object. The template of data storage structures and methods that can be used on them is often defined in a separate file, so we'll imagine that each of these methods are written in a separate file. Of course, this file does not actually exist anywhere in the `OSCAAR` package and the following pseudocode will not work if you try to run it unless you wrote out the definitions for this `car` object.

The first thing we'll do with the `car` object is *initialize* it, which means to assign an *instance* of the object to variable. We'd do that like this in Python:

```
>>> batmobile = car() ## Initialize an instance of the car object
```

Then we can store values into the template that's already defined for the `car` object. For example, let's imagine that the `car` object has a property that stores the number of tires that this particular `car` object has that we'd like to set using one of the `car` object's *methods*. A *method* is a function that works on values within an *object*. We could return the number of tires using the `get_number_tires()` method and set it using the `set_number_tires()`. In our example, we could do

```
>>> N_tires = 4
>>> batmobile.set_number_tires(N_tires) ## Use a method on a object by doing object.method()
>>> print batmobile.get_number_tires()
4
```

The methods for each object have access to the data that you've stored in that object. So from here, we could use the `get_number_axles()` method that would look up the number of tires that we stored in the `batmobile` instance of the `car` object using the `set_number_tires()` method, which divides that number of tires by two:

```
>>> Naxles =  batmobile.get_number_axles()
>>> print Naxles
2
```

Objects are used in `OSCAAR` to help organize data and execute functions on those data without asking the user to name and manage arrays that handle all of the fluxes, the uncertainties on those fluxes, the stellar centroid positions, etc. This helps to keep your code clean to keep things modular. For instance, let's say that Batman blew out a tire, so his number of tires is now 3. The `get_number_axles()` only divides the number of tires by two, and which will return 1.5 axles now. We know he still has at least two axles though since he has more than two tires, so we'll write and use a more sophisticated method `get_number_axles_special()` that will divide the number of tires by two and round up to the nearest integer. Then we'd only need to change the call to the

`get_number_axles()` method to `get_number_axles_special()`, which is quick and easy to implement and experiment with.

One of the goals of `OSCAAR` is to have lots of different methods for each task so that you can try different ways of reducing your data without having to change out large chunks of code. If you used `trackSmooth()` to find stellar centroids (covered in Section 4.2), but thought a routine that fits two-dimensional gaussians to the images of the stars would work better, you could write a method for the `dataBank` class that would take the same input parameters as `trackSmooth()` and return the same outputs. Then all you would have to do is change the word "trackSmooth" to the name of your gaussian fitting method in the `differentialPhotometry.py` script to use the new algorithm. All of the other parts of the code could stay the same and it would be easy to change back to `trackSmooth()` if you wanted to.

4.3.2 `dataBank.calcMeanComparison()`

In order to make a transit light curve, we want to divide each flux measurement of the target star by the flux of a star that has no intrinsic flux variations, and ideally a star of similar spectral type to the target. How are we to know which star is the right one to use as a comparison star? To make matters worse, your detector may not be evenly responsive over the whole of the imaging surface, so stars that fall near or far away from the target star on the detector may be better to use as comparisons than others.

We often take the mean of a bunch of comparison stars so that they can collectively make a "composite" comparison star, or "mean" comparison star, in the wording of the name of this algorithm. If one star varies a little bit or falls on a worse part of the detector, it may not introduce significant variability into the mean of the comparison stars since it gets diluted by a the non-varying fluxes of the other stars.

Let's consider the case with two comparison stars. Ideally, we would have to perfect comparison stars imaged by a perfect detector. Their spatial positions on the CCD would not affect the measurements of their fluxes and they would have no intrinsic flux variations. In this case, the vector of mean comparison star fluxes $\mathbf{f}_{\bar{C}}$, (though unnecessary) could be produced by taking the fluxes of each comparison star $\mathbf{f}_{C,0}$ and $\mathbf{f}_{C,1}$, and taking the average:

$$\mathbf{f}_{\bar{C}} = a \cdot \mathbf{f}_{C,0} + b \cdot \mathbf{f}_{C,1} \tag{2}$$

where a and b here are scalar multiplier coefficients for the vectors of fluxes, with $a = b = 0.5$ in this case. We would then proceed to calculate the light curve $\mathbf{f}_{relative}$ by simple division

$$\mathbf{f}_{relative} = \frac{\mathbf{f}_T}{\mathbf{f}_{\bar{C}}} \tag{3}$$

where $\mathbf{f}_T$ is the vector of target fluxes.

Now let's say that the image of the first comparison star fell on a few dead pixels on the detector, or that it has some slight intrinsic variability that does not make it a good comparison star. If we were to use Equation 2 with $a = b = 0.5$ we'd be considering the first and second comparison stars with equal weight in making the mean comparison star. A more useful "mean" comparison star wouldn't use the mean of the two comparison stars, rather it would take some other linear combination of the fluxes with different coefficients to make the composite comparison star $\mathbf{f}_{\bar{C}}$. In this case, we would multiply the first comparison star fluxes $\mathbf{f}_{C,0}$ by a small coefficient, perhaps $a = 0.2$, and since $a + b = 1$, $b = 0.8$. This will "put a heavier weight on" the fluxes of the second comparison star in the calculation of the mean comparison star, and should produce a better light curve.

The `dataBank.calcMeanComparison()` algorithm takes the fluxes of each normalized comparison star with fluxes $\mathbf{f}_{C,i}$ and tries different weights on each star. It will minimize the difference between the out-of-transit

portions of the target star fluxes and the mean comparison star fluxes by varying a set of coefficients $\mathbf{c}_i$ that get multiplied by each comparison star flux such that $\Sigma_i c_i = 1$.

$$\begin{aligned} \mathbf{f}_{\bar{C}} &= c_0 \cdot \mathbf{f}_{C,0} + c_1 \cdot \mathbf{f}_{C,1} + c_2 \cdot \mathbf{f}_{C,2} + ... \quad (4) \\ &= \Sigma_i \; c_i \cdot \mathbf{f}_{C,i} \quad (5) \end{aligned}$$

The vector of the uncertainties of the composite comparison star fluxes $\sigma\mathbf{f}_{\bar{C}}$ is then

$$\begin{aligned} \sigma\mathbf{f}_{\bar{C}} &= \sqrt{(c_0 \cdot \mathbf{f}_{C,0})^2 + (c_1 \cdot \mathbf{f}_{C,1})^2 + (c_2 \cdot \mathbf{f}_{C,2})^2 + ...} \quad (6) \\ &= \sqrt{\Sigma_i \; c_i^2 \cdot \mathbf{f}_{C,i}^2} \quad (7) \end{aligned}$$

Of course, the out-of-transit exposures are the only ones used in this calculation because there is an important intrinsic flux variation in the target star during transit (that we're trying to see!). If we ran this regression technique to compare the target star to comparison stars on exposures taken during the transit, the algorithm would find the linear combination of target stars that best match the variability of the target star, and it would effectively seek to cancel out the transit. This is why we enter the ingress and egress times into `OSCAAR` (see Section 3.3.7), so that we can identify which exposures to use in these comparisons.

This method also effectively cleans up after us if we choose bad comparison stars, as laid out in Section 3.1. Let's say we chose a dim star that had a brighter nearby companion. The tracking algorithm may be able to accurately find the centroid of the dim star that we intended to use as a comparison star for a few exposures or maybe even for half of the exposures, but it may eventually "see" the brighter star nearby and start tracking its centroid instead. When this occurs, the flux measured for this comparison star will suddenly jump up discontinuously at the exposure when the centroid of the brighter star was found. The discontinuous flux jump in fluxes of this comparison star will be very dissimilar to continuous, telluric variations that effect the target star on a night with decent weather. The `dataBank.calcMeanComparison()` algorithm will find that only very small coefficients for the flux of this comparison star will produce a good composite comparison star, essentially giving this star no weight in the composite comparison star, counting it out.

4.3.3 `dataBank.computeLightCurve()`

We calculate the light curve vector $\mathbf{f}_{relative}$ by simple division of the flux of the target star $\mathbf{f}_T$ by the flux of the composite comparison star $\mathbf{f}_C$, generated using `dataBank.calcMeanComparison()` (see Section 4.3.2).

$$\mathbf{f}_{relative} = \frac{\mathbf{f}_T}{\mathbf{f}_{\bar{C}}} \quad (8)$$

The errors for each light curve flux are calculated by propagating of uncertainties. If $\sigma\mathbf{f}_{\bar{C}}$ is the vector of uncertainties in the composite comparison star fluxes and $\sigma\mathbf{f}_T$ is the vector of uncertainties in the fluxes of the target star, the uncertainties in the light curve fluxes $\sigma\mathbf{f}_{relative}$ are

$$\sigma\mathbf{f}_{relative} = |\mathbf{f}_{relative}| \sqrt{\left(\frac{\sigma\mathbf{f}_T}{\mathbf{f}_T}\right)^2 + \left(\frac{\sigma\mathbf{f}_{\bar{C}}}{\mathbf{f}_{\bar{C}}}\right)^2}. \quad (9)$$

4.4 `oscaar.scaleFluxes()`

5 Troubleshooting

If you notice a problem in OSCAAR or can't get something to function properly, feel free to submit an issue on our Issue Tracker on GitHub.

6 Contributing to OSCAAR

This version of OSCAAR was made by a very small group people, and we're proud of the work we've done. But OSCAAR can still be improved and we need your help! OSCAAR is used around the world by amateurs and professionals alike, and in order to keep up with the demands of providing a user-friendly differential photometry code for an international audience, we'd love to have your help if you can code in Python, or provide any feedback at all. If you'd like to help but don't know where to begin, please feel free to contact us at oscaarteam@gmail.com.

OSCAAR started as one of Brett Morris's independent undergraduate research projects at the University of Maryland, and since then has fueled independent research projects for several other undergraduates. If you are an undergraduate studying astronomy, physics or computer science and would like to contribute to oscaar, we encourage you to reach out to us. We'd love to work together!

We keep the source code on GitHub, a popular open source code repository site. There you can find the code in its most up-to-date (alpha, beta, and stable) form, the Issue Tracker where known issues are logged and new issues or comments can be posted.

7 Acknowledgements

OSCAAR has come a long way from the first 1,000 lines of code that Brett Morris wrote in 2011. It could not have gotten there without the help of the following colleagues: Professor Drake Deming, Dr. Avi Mandell, Daniel Galdi, Sam Gross, Luuk Visser, Harley Katz, Elizabeth Warner, Dr. Alberto Bolatto.