Draft version December 5, 2022
Typeset using LaTeX **twocolumn** style in AASTeX63

Pipeline Progress Fall 2020

ABSTRACT

This paper overviews our progress this quarter developing a transient detection data pipeline which employs both difference image subtraction and light curve analysis techniques to identify and categorize transient light source candidates, specifically transients which are characteristic of directed energy communications. I am one contributor to this large undergraduate effort spearheaded by Dr. Philip Lubin. I will pay specific attention to what I have been integral to developing and how it relates to our science objective of real time image processing.

1. INTRODUCTION

Many sections are adapted from our paper which will tentatively appear in the literature as Local Galactic Transient Survey Background. The Search for Extraterrestrial Intelligence (SETI) seeks to discover extraterrestrial communications using available communication methods. Historically, this search has focused on radio frequencies, which were well understood as means of communication when SETI efforts began in the 1960s. Recent advances in laser technology suggest that lasers may be an effective means of interstellar communication. Thus, examining optical and near infrared frequencies for laser communications is a natural extension of SETI efforts, especially because few SETI searches have examined these frequencies.

While the optical and near-infrared (IR) frequencies present many opportunities for SETI efforts, it also presents a new set of challenges. The most significant issue is that the optical and near-IR bands contain significantly more noise, requiring more processing effort to identify whether sources as intelligent.

In our pipeline, we adapt techniques used to identify naturally occurring transient sources, such as variable stars and supernovae, to identify optical and near-infrared directed-energy sources.

Before looking for a directed energy (DE) signal, we must be able to describe and understand what we are looking for so that we know when we have found it. Lubin (2016) anticipates the possibility of civilizations attempting to communicate by using DE signals and attempts to quantitatively and qualitatively describe what these signals might "look like," depending on the distance of the civilization and the development of their technology. Lubin concludes that it is not unreasonable for an extraterrestrial source to be detected at even vast distances using modest searches. From a galaxy like Andromeda, the signal would appear to be a transient but non-periodic (in human time scales) foreground point source.

In particular, Lubin predicts that the apparent flux in Wm^{-2} of a laser emitted from distance L is given by the equation

$$F = \frac{\xi P}{L^2 \Omega} = \frac{\xi F_e \epsilon_c 10^{4S}}{4L^2\lambda^2}. \tag{1}$$

Here, $\xi = \frac{\lambda}{hc}$ for a given wavelength (λ), Plank's constant is h, and the speed of light is c. Power (P) = $F_e \epsilon_c 10^{2S}$, for a given solar illumination (F_e = 1400 W/m^2), conversion efficiency of solar power to laser power, ϵ_c, is $\mathrm{eff}_{pv} * \mathrm{eff}_{de}$, and for a given civilization of class S. Ω is solid angle and equals $4\lambda^2 10^{-2S}$ sr.

Furthermore, he finds that the flux for a given civilization class, based on his definition, is

$$\frac{P}{\Omega} = \frac{F_e \epsilon_c 10^{4S}}{4\lambda^2} = \frac{350 \epsilon_c 10^{4S}}{\lambda^2}. \tag{2}$$

Thus, assuming a class 4 civilization transmitting from Andromeda galaxy (M31), we take $S = 4$, $\epsilon_c = 0.5$ (also from Lubin's paper), λ to be between 400 nm and 700 nm (optical wavelengths), and $L \approx 2.56 \pm 0.08$ Mly (McConnachie et al. (2004)). These values give us an approximate upper and lower bound for the flux: $F_{\mathrm{lower}} = 2 * 10^4 \ \frac{\mathrm{photons}}{\mathrm{m^2 s}}$ to $F_{\mathrm{upper}} = 4 * 10^4 \ \frac{\mathrm{photons}}{\mathrm{m^2 s}}$.

From these flux values and presuming an R band filter, a prospective magnitude for a signal from the Andromeda galaxy is 16, which is well within the seeing capabilities of the our telescopes and the pipeline's detection ability. However we do not currently use filters for our observations, so a source may be easier to observe. If we take M31 to have a uniform distribution,

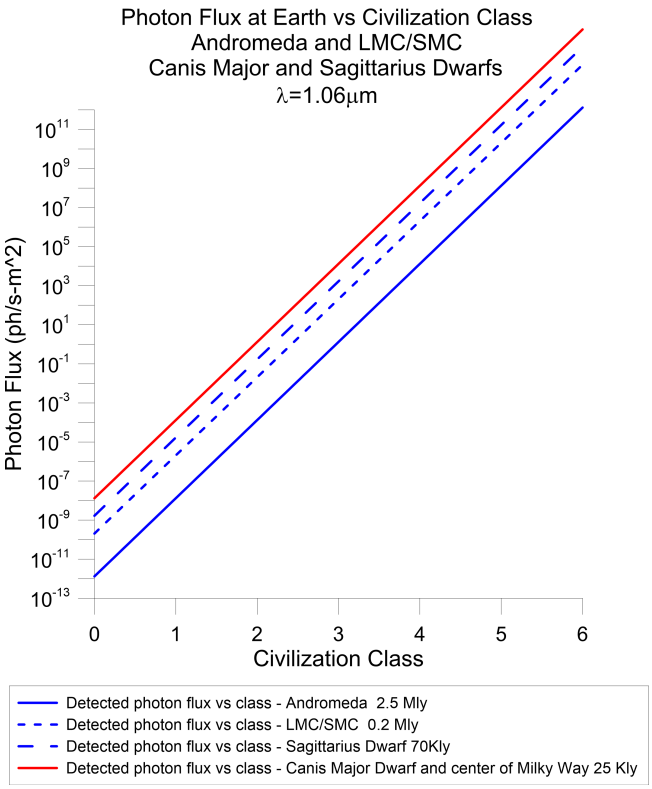

Figure 1. Photon flux incident at Earth emitted by various civilization classes at wavelength 1.06 μm. The solid blue line is M31. The solid red line is Canis Major Dwarf and the center of the Milky Way. The short dashed blue line is the Large and Small Magellanic Clouds. The long dashed blue line is Sagittarius Dwarf.

then its surface brightness is 22. Considering the predicted dwell time, any such source should be detectable with the 0.4 meter telescopes we employ (other than perhaps a source in the center of the galaxy).

Figure 1 summarizes the photon flux versus Civilization class for intragalactic and some nearby intergalactic galaxies.

Dwell time, τ, is the dwell time of laser on Earth and is given by

$$\tau = \frac{2L\lambda}{10^S V_t}, \tag{3}$$

where and V_t is transverse speed of the transmitter relative to Earth.

Again, taking a $S = 4$ civilization class in Andromeda where the distance, L, is approximately 2.54 ± 0.11 Mly. Optical band wavelengths puts λ between 400 nm and 700 nm. A typical transverse speed according to Lubin's paper is $V_t = 100\frac{\text{km}}{\text{s}}$ to $1000\frac{\text{km}}{\text{s}}$.

Thus, the lower bound for dwell time is $\tau_{\min} = 2 \times 10^6 s$ and the upper bound of dwell time is $\tau_{\max} = 4 \times 10^7 s$. A plot of dwell time versus distance for various

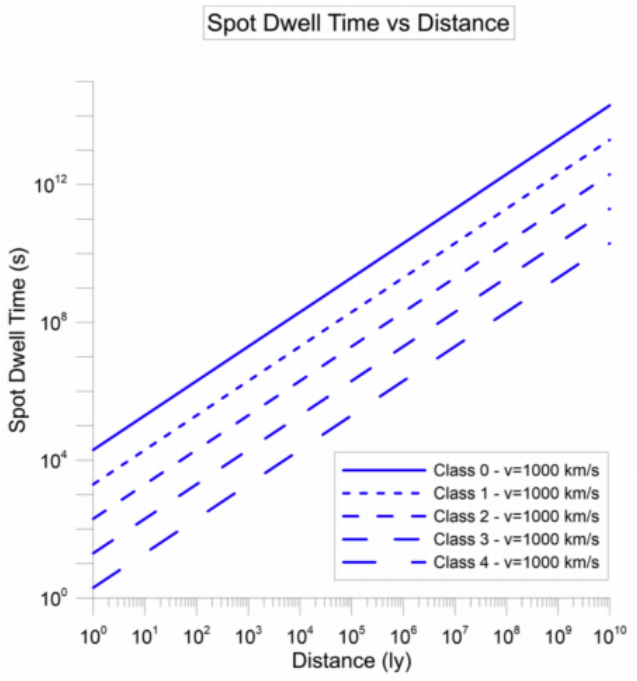

Figure 2. Spot dwell time vs distance. Lines plotted for various civilization classes. Figure reproduced with permission from Lubin (2016).

civilization classes can be found in Figure 2. Given that these bounds are orders of magnitude higher than our science images, having between 10 s to 1000 s integration times, it is quite feasible for our telescopes to capture any such signals. Dwell time should only become an issue with high-class civilizations in our own galaxy, as it shortens with lower distances, shorter wavelengths, and higher transverse velocities.

Lubin also predicts a signal to noise ratio (SNR):

$$\frac{S}{N} = \frac{FA_\epsilon\tau}{[N_R^2 + \tau n_t^2]^{1/2}}. \tag{4}$$

Here F is flux from target in $\gamma s^{-1} m^{-2}$. $A_\epsilon = A\epsilon Q_e$ is the effective telescope area including transfer efficiency and quantum efficiency in $m^2 e^- \gamma^{-1}$, where A is the telescope area, ϵ is the telescope transfer efficiency, and $Q_e(\lambda)$ is the quantum efficiency of the detector in $e^-\gamma^{-1}$. τ is the integration time in s. N_R is the readout noise in e^-.

Since $n_t = \sqrt{FA_\epsilon + i_{DC} + F_B A_\epsilon \Omega}$ is the signal, dark current, and background noise in $e^- s^{-1/2}$, we can rewrite the above equation as:

$$\frac{S}{N} = \frac{FA_\epsilon\tau}{[N_R^2 + \tau(FA_\epsilon + i_{DC} + F_B A_\epsilon \Omega)]^{1/2}}. \tag{5}$$

Here Ω is the solid angle of pixel in sr. F_B is the flux per solid angle from all background sources integrated over

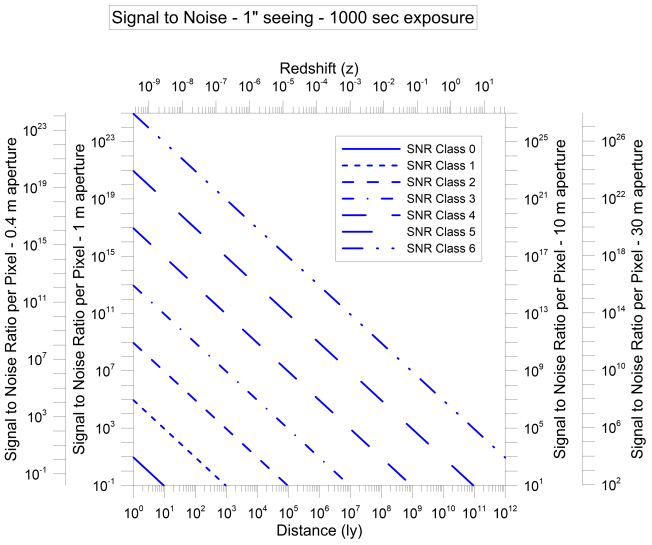

Figure 3. Signal to noise vs distance for various apertures. Using the scale for the 0.4m aperture, we can determine what minimum civilization class can be detected for a given distance. The integration time only affects the noise not the signal in the intelligent targeting assumption. Our typical integration time per image is much shorter which makes the SNR even higher. We use 1000 sec for the integration time here to emphasize that even long integration (exposure) times yield large SNR. See Time to SNR figure below for additional discussion.

bandwidth in $(\gamma s^{-1} m^{-2} sr^{-1})$. And i_{DC} is the detector dark current in $e^- s^{-1}$.

The term FA_ϵ in the denominator is noise produced by the signal glow and should apply only if SNR is computed relative to the pixel source. Thus, we can omit this term reducing our equation to:

$$\frac{S}{N} = \frac{FA_\epsilon \tau}{[N_R^2 + \tau(i_{DC} + F_B A_\epsilon \Omega)]^{1/2}}. \quad (6)$$

Therefore, the time for a desired SNR, τ, is

$$\tau = \frac{S_N^2 n_t^2 \pm \sqrt{S_N^4 n_t^4 + 4F^2 A_\epsilon^2 S_N^2 N_R^2}}{2F^2 A_\epsilon^2} \quad (7)$$

Now, letting $N_t = [N_R^2 + \tau n_t^2]^{1/2}$ for the noise in e^-, we can rewrite:

$$\tau = \frac{S_N^2 n_t^2}{2F^2 A_\epsilon^2} \left(1 + \sqrt{1 + \frac{4F^2 A 2_\epsilon^2 N_R^2}{S_N^2 N_t^4}}\right) \quad (8)$$

Figure 3 summarizes the results of SNR as a dependent of several apertures resulting in a scaling factor that Noise has two components: readout noise and time-dependent noise. For short integration times, readout noise dominates, but, at longer integration times, the time-dependent part dominates. The transition between these two states occurs at transition time τ_c and is calculated with the following:

$$\tau_c = \frac{N_R^2}{n_t^2} = \frac{N_R^2}{FA_\epsilon + i_{DC} + F_B A_\epsilon \Omega} = \frac{N_R^2}{i_{DC} + F_B A_\epsilon \Omega} \quad (9)$$

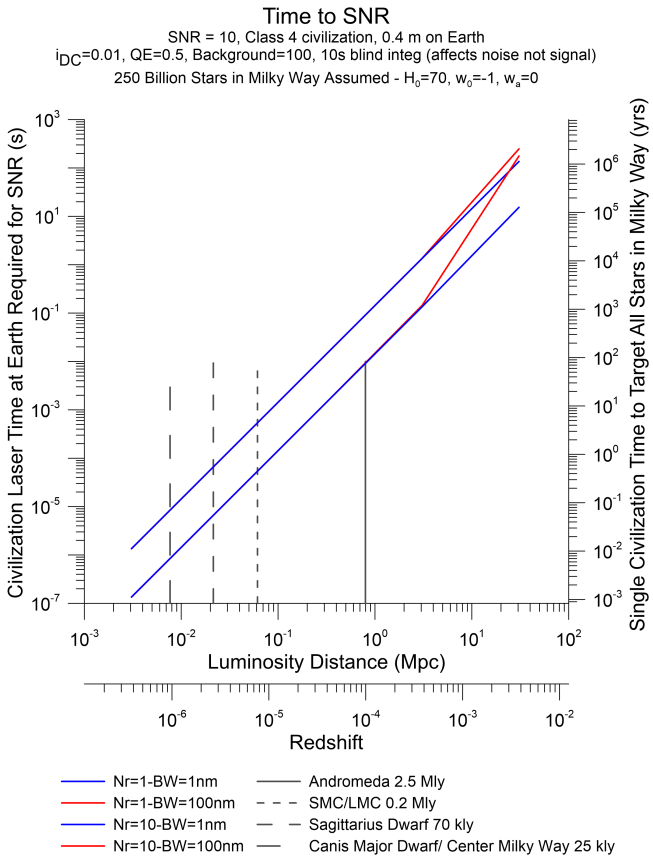

Figure 4. Time to SNR vs Luminosity Distance. For a civilization class 4 laser, detection by a 0.4m telescope on Earth for the desired SNR ratio in several nearby galaxies falls within about 10 ms of exposure time.

From this paper, the Trillion Planet Survey (Stewart & Lubin (2017)) was conceived, where 0.8m telescopes from the Las Cumbres Observatory were used to look for continuous wave lasers in the optical frequencies. However, due to problems with its difference imaging pipeline, this Trillion Planet Survey was not able to be completed. Instead, we now present our improved and verified methodology and pipeline in a new SETI type survey.

2. METHODS

2.1. *LCO Data Collection*

Data is taken using the LCOGT network's system of 0.4 m telescope arrays with a FOV of 29.2×19.5 arcmin

Brown et al. (2013). Observations are focused on the Andromeda galaxy due to its large size, favorable relative angle, density of stars, and close proximity. Based on our calculations in the previous section (1), we expect that a source from a class 4 civilization from this galaxy would be around 16 magnitude or brighter. Thus, we should expect that a source from a class 4 civilization from this galaxy would be around 16 magnitude or brighter. Thus, we should have a high probability of detecting any such source, especially considering our magnitude detection ability outlined in section 2.9. Since LCOGT's 0.4 m telescopes have a field of view of 19 by 29 arcseconds, we divided the galaxy into a mosaic of observable sections, each of which spans the smaller field of view depicted in Figure 5.

Focus is on the sectors of the mosaic containing the galaxy as shown in the figure. Sample data for section 33, 38, and 32 can be seen in Figures 6, 7, and 8 respectively. This sample of sectors show varying star density of the tiled-mosaic M31 as we take images closer to the center of the galaxy. Here section 38 is near the peripheral of M31, section 33 is nearer the center, and section 32 is the closest of the three to M31's dense center.

Data for this project was historically collected in large batches over a short period of time for all sectors of the galaxy. We imaged batches of 3-5 sectors once approximately every 14 days, building a backlog of images over consistent intervals so we can detect transients through comparison of data taken over different parts of their transit. To date, we have 5.3 Tb comprising over 35,000 images and are continuing to take more over these specified intervals.

Each observation request takes 100 images at a time with 10 second integration times. These images are taken without filters so detection of a transient in any of the optical and near infrared wavelengths is possible. Potential transient candidates can then be imaged with filters to determine more details about the nature of the transient.

In addition to M31, we are also imaging the Large and Small Magellanic Clouds when M31 is not visible to LCOGT since, by the same selection criteria as used for Andromeda galaxy, these clouds are close targets with a number of stars that can be easily imaged.

2.2. *Broida Rooftop Observatory Data*

In addition to LCO data, we are constructing an observatory atop the physics building (Broida Hall) to give undergraduates hands-on experience aligning telescopes and taking data. This observatory is built using hardware generously donated by LCO along with automation hardware and a low-cost tensor processing unit that can

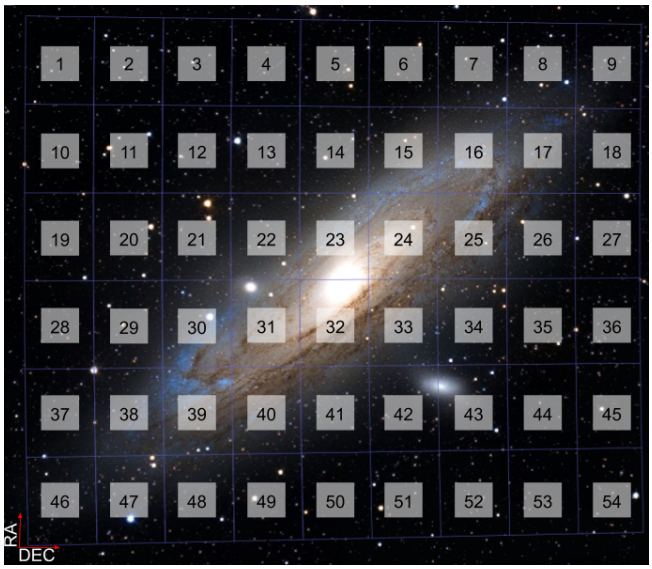

Figure 5. Andromeda Mosaic: Sections of M31 imaged by the LCOGT network's 0.4m arrays are 7, 8, 11, 13, 14, 15, 16, 17, 21, 23, 24, 25, 26, 30, 31, 32, 33, 34, 38, 40, 41, 42, and 47.

Figure 6. Section 33 of M31. Taken at the LCOGT node at Haleakala Observatory on 2019 August 18 using an integration time of 100s. This image is the result of the raw image applied to the LCO's BANZAI pipeline and is typical of the science images given to our DaLEC pipeline.

quickly process these 6280×4210 pixel (26 Mpixel) images using our pipeline. Our QHY 268 has a lower angular resolution with 3.88 arcsec^2 pixels and larger pixel count creating a much larger FOV of $3.4\,\mathrm{deg} \times 2.3\,\mathrm{deg}$.

Due to the image size and un-optimized memory utilization, GPU accelerated subtraction of a single QHY images may require up to 60 Gb of VRAM which may be prohibitive even to capable computing systems. This constraint is currently being revised by Dr. Lei Hu at the University of Science and Technology of China which should drastically reduce VRAM utilization allowing low

Figure 7. Section 38 of M31. Taken at the LCOGT node at McDonald Observatory on 2019 October 18 using an integration time of 60s. This image is the result of the raw image applied to the LCOGT network's BANZAI pipeline and is typical of the science images given to our DaLEC pipeline.

Figure 8. Section 32 of M31. Taken at the LCOGT node at McDonald Observatory on 2019 July 19 using an integration time of 100s. This image is the result of the raw image applied to the LCOGT network's BANZAI pipeline and is typical of the science images given to our DaLEC pipeline.

cost tensor processing to be feasible. A full discussion of this issue can be found in 2.3. If the memory optimizations are insufficient, processing images in slices can provide the remaining needed optimization. An intelligent slicing algorithm that divides images into slices of the correct size for the available hardware would allow GPU subtraction of any $n \times m$ image with any amount of VRAM. However, there is likely a decrease in the quality of results due to slicing since smaller images introduce more edge effects, and may create alignment and source detection problems. Therefore, our efforts have currently focused on mitigating the need for this slicing step which serves as a backup solution.

2.3. *Difference Image Analysis*

Figure 9. A raw image taken from the Broida Rooftop showing Andromeda and Triangulum along with many background stars.

We use a variety of trusted astronomical science data programs to create our differencing imaging pipeline using a Python framework. These include Source Extractor Python (SEP) (Barbary (2016), Bertin (2017)), Optimal Image Subtraction (OIS) Adaptive Bramich method (Beroiz et al. (2020), Miller et al. (2008)), Astroalign (Beroiz et al. (2020)), NumPy (Harris et al. (2020)), as well as the LCOGT network's own BANZAI pipeline architecture for image preprocessing and identification (McCully et al. (2018)). We give an overview of our processing procedures in the flowchart in Figure 10 as well as in section 3.2.

The general structure of our pipeline is as follows. The images are first taken using one of the LCOGT network's 0.4m telescopes and then preprocessed according to their BANZAI pipeline. Each of these science images are then sent into our DaLEC (Differencing and Light Exposure Curve) pipeline for further processing, which includes difference image analysis and database cataloguing.

The DIA functionality of the DaLEC pipeline is composed of four primary operations: Align, Combine, Subtract, and Extract. These steps are diagrammed in Figure 10. LCOGT's preprocessing and each of the four steps will be discussed in turn.

2.4. *LCO BANZAI Preprocessing*

Preprocessing occurs in the LCOGT network's internal BANZAI pipeline. Raw images taken from their observatories are run through a series of steps to produce science quality images. The processes applied to the images are overscan subtraction, gain, bias subtraction, flat field correction, source detection, and astrometry. See the LCOGT network's documentation on BANZAI for more detail (McCully et al. (2019)). Also, see Figure 11 for an example of a raw image from before it was run

through the BANZAI pipeline and compare to Figure 8 to see the changes preprocessing makes.

2.5. *Broida Rooftop Preprocessing*

After investigation the BANZAI pipeline with Joe Politi, we decided the BANZAI pipeline was prohibitively restrictive for our data. The BANZAI pipeline expects images exactly as LCO creates them. Since it is designed specifically for LCO data, it would require writing a prepreprocesing step that reformats our data to match BANZAI's needs. This idea was dropped after the installation process took multiple hours and after finding their documentation difficult to an outsider with the idea that it is better to utilize and develope something we understand completely than adapt our work to fit into a black box.

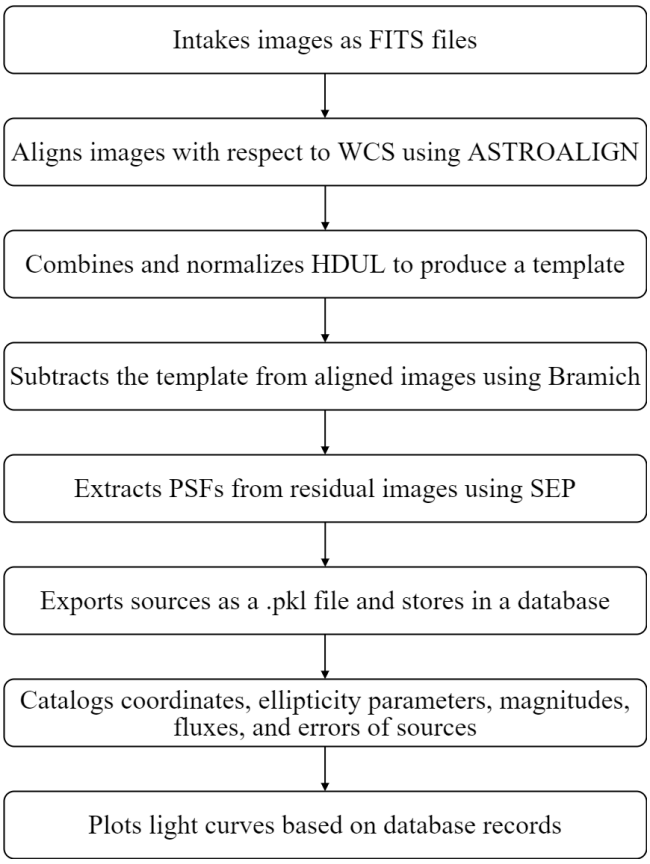

Figure 10. Flowchart of the DaLEC data pipeline. Science (CCD) images are loaded into the pipeline as FITS files. They are aligned using ASTROALIGN to a reference image that is either picked automatically by the pipeline or specified by the user, then combined into a template image. Reference image analysis is performed and variable sources are extracted using SEP. The subtracted images are stored and the sources are put into our database with coordinates, ellipticity, magnitude, and flux.

Figure 11. Raw image of section 32 of Andromeda before the preprocessing step. Once the LCOGT network's BANZAI pipeline has been applied, we obtain the image in Figure 8. Notice the removal of the dark spot circled in the raw image as well as the increase in detail between the preprocessed image and this raw image.

```
alex@starmaker:~/pits/pi-in-the-sky/pits$ pits --help
Usage: pits [OPTIONS] COMMAND1 [ARGS]... [COMMAND2 [ARGS]...]...

  Chains together pipeline commands.

Options:
  --help  Show this message and exit.

Commands:
  alignment  rotates frames and preprocessing image alignment
  calibrate  calibrates the image with bias images and flatfielding
  read       Takes a directory containing fits files and returns them as...
```

Figure 12. Help menu of the PITS preprocessing pipeline showing how to invoke the pipeline with brief descriptions of the read, alignment, and calibrate commands

Image processing via the DaLEC pipeline is very straightforward as it can simply be evoked by typing sdi into the command line, followed by the desired modules. Preprocessing was not command line integrated and could not be done in this way. To select images for the pipeline, the user had to manually write directory names into the scripts themselves. Utilizing click, the directories can now be specified from the command line itself. This has been a slow process of reverse engineering a computer scientist's uncommented code and building my own model (Ronacher (2022)). After many weeks of effort, I can now preprocess images entirely from the command line. Shown below is the help menu of the new pipeline I authored.

2.6. *Image Alignment*

In order to isolate differences over time through DIA, science images must be positionally and photometrically (see **??**) aligned. We perform positional alignment by using AstroAlign (Beroiz et al. (2020)), a Python package created for this purpose. AstroAlign deduces an appropriate align transformation by identifying and com-

paring asterisms in the reference image and in the image to be aligned. The random sample consensus algorithm (RANSAC) is applied to eliminate the influence of outliers and asterism mismatches. Finally, AstroAlign returns a matrix transformation which is applied to the science image to produce a positionally aligned image. Figure 13 is an example of the image before and after positional alignment.

To accelerate this process, we employed a multiprocessing approach. This allows multiple images to be aligned in parallel, vastly accelerating the process. While in theory this can accelerate processing by a factor of the number of CPU cores (72 for our hardware), in practice we saw an increase by a factor of 10.

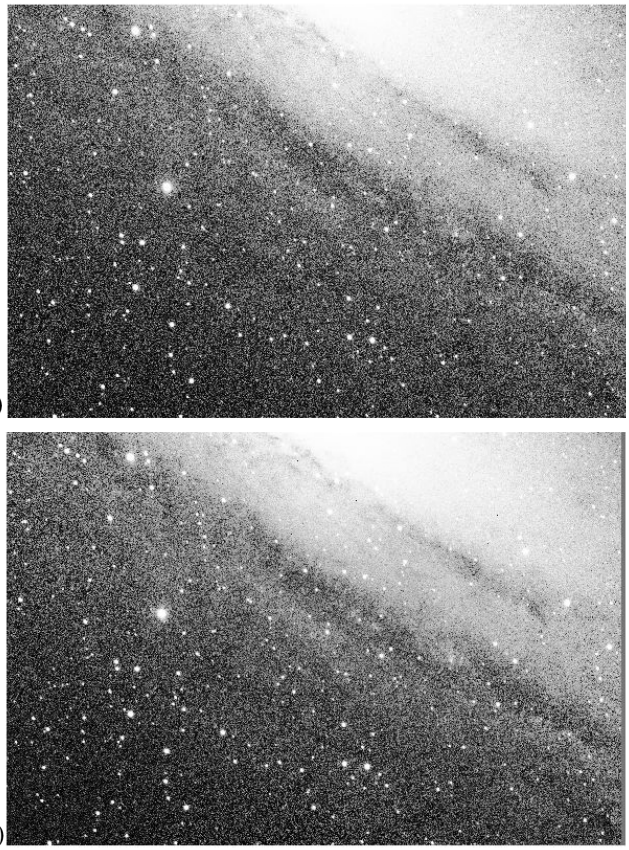

Figure 13. Single image of M31 section 24 that has only gone through preprocessing in the BANZAI pipeline (a) verses the same image after the align step (b). The small vertical line on the right side of (b) is an artifact of our alignment process.

2.7. *Image Combination*

Once the images are aligned, the pipeline moves on to the combine step which combines all images in a single input dataset (for which all images are of the same section of the sky) thus forming a template image for later use. The algorithm does this by extracting the value of each pixel from all the input images, then taking the median of each pixel. The median pixel value is then used as the value for the corresponding pixel in our template image. This method creates a template image with a substantially higher signal to noise ratio than using the mean pixel value, as shown in Drake et al. (2009), which is useful for later subtraction. See Figure 14 for an example of the process.

Figure 14. Single image of M31 section 24 (a) versus a combined image made from 100 single images (b)

2.8. *Subtract*

Before a science image and the template image can be subtracted, they must be photometrically aligned. This alignment is done by applying a convolutional kernel to the better-seeing template image so that its point-spread function matches that of the science image. The appropriate convolutional kernel to use is identified by applying optimal image subtraction techniques, which vary kernel parameters to minimize the differences between the two images. We chose to use a space-varying delta basis kernel as proposed in Miller et al. (2008).

We selected a delta basis kernel because of its increased effectivity on real datasets as compared to Gaussian basis kernels, as shown in Sánchez et al. (2019).

A space-varying application of the delta-basis kernel is especially important for our uses, as many of our images contain elements of M31's galactic structure, dramatically changing photometric conditions in different regions of the image. A space-varying kernel is multiplied by a polynomial function of the pixel coordinate that the convolution is applied to, allowing the kernel to adapt to different photometries in different regions of the image.

We implemented a space-varying delta-basis kernel fit by using the Adaptive Bramich method in the OIS Python package (Beroiz et al. (2020)). Because of the large number of parameters that need to be fit and the time-consuming convolution process, fitting an Adaptive Bramich kernel took more than 40 minutes for one of our 2042×3054 science images, with over 200,000 calls to the `multiply_and_sum` convolution function. In order to accelerate the process, we implemented portions of the OIS package using the CUDA toolkit, allowing it to be massively parallelized on Graphics Processing Units. This brought the computing time for each image down to about 10 minutes per image; long, but bringing dataset turnaround times from weeks to days.

The Adaptive Bramich method still requires orders of magnitude more computation time than spatially invariant approaches such as the Bramich method. We found that the increased efficacy of Adaptive Bramich more than justifies the longer computation times required to apply it.

Across the board, the residuals from images subtracted using Adaptive Bramich have a lower standard deviation as well as mean that is closer to 0 in all cases, but unsurprisingly more pronounced for images with larger amounts of galactic structure. In data with minimal galactic structure, Bramich subtraction yielded an average standard deviation of 16.8 compared to an average standard deviation of 15.1 for Adaptive Bramich. For data with low to moderate galactic structure present, the two subtraction methods produced average deviations of 41.99 and 35.04. In data containing galactic core, Adaptive Bramich produced better results having a standard deviation of 241.188 compared to 247.89 for regular Bramich. These numerical values are representative of the outcomes for masked data. Moving forward, the implementation of a more robust point-source mask coupled with Adaptive Bramich will improve the outcomes of the subtraction step of the pipeline.

See Figure 15 for a visualization of the subtract process on section 24 of M31 as well as 16 for the associated histogram of the residual image. Comparing 16 to 17, we can see that the former is a much better subtraction as the residual noise takes on a much more Gaussian distribution with a tighter standard deviation and mean value that is closer to 0. This result is characteristic of all tests we have run comparing Bramich and Adaptive Bramich.

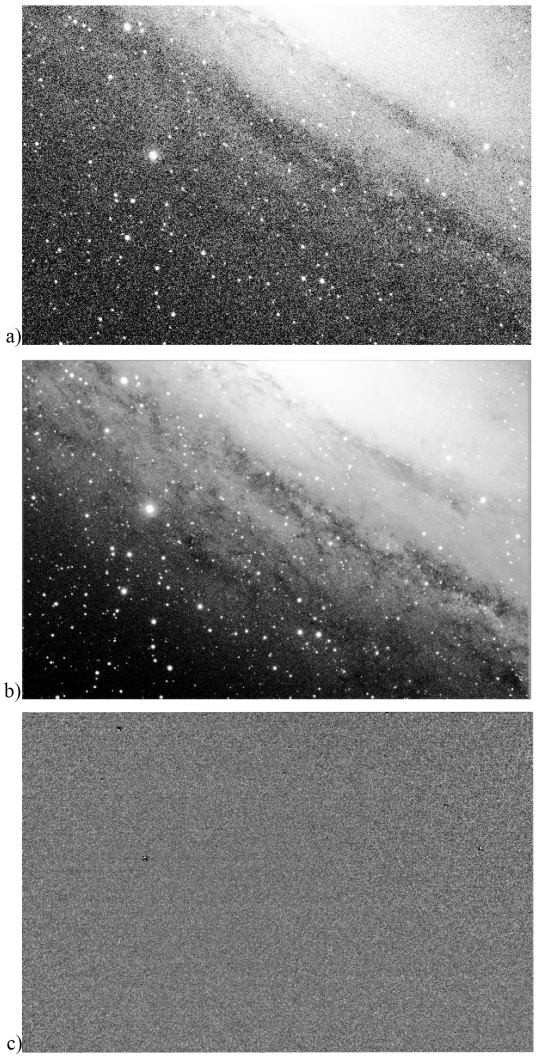

Figure 15. These are 3 images of M31 section 24 which showcase the DALEC data pipeline's image subtraction. (a) is a single image of the section from which the combined template image (b) is subtracted to achieve the residual image (c).

2.9. *SFFT Subtract*

We implemented the SFFT image subtraction algorithm developed by Dr. Lei Hu. To quote his work, "SFFT uses δ-function basis for kernel decomposition, and the image subtraction is performed in Fourier Space. This brings about a remarkable improvement of computational performance of about an order of magnitude compared to other published image subtraction codes. SFFT can accommodate the spatial variations in wide-field imaging data, including PSF, photometric scaling, and sky background. However, the flexibility of the δ-function basis may also make it more prone to overfitting. The algorithm has been tested extensively in real astronomical data taken by a variety of telescopes. Moreover, the SFFT code allows for the spatial variations of the PSF and sky background to be fitted by spline function"Hu et al. (2022).

SFFT subtraction was written for "to use sep for extracting the sky background rapidly". To utilize SExtractor, we currently write out temporary fits files that are deleted at the end of the pipeline. Subtraction using SFFT is highly intensive on VRAM. Currently processing 26 Mpixel images requires 60 Gb of VRAM. Our chosen Jetsen Nano tensor processing unit has 4 Gb of VRAM. Therefore, the memory requirements must be reduced by a factor of 15 in order to make sfft subtraction of 26 Mpixel images possible without employing the slicing techniques discussed in 2.2.

2.10. *Extract*

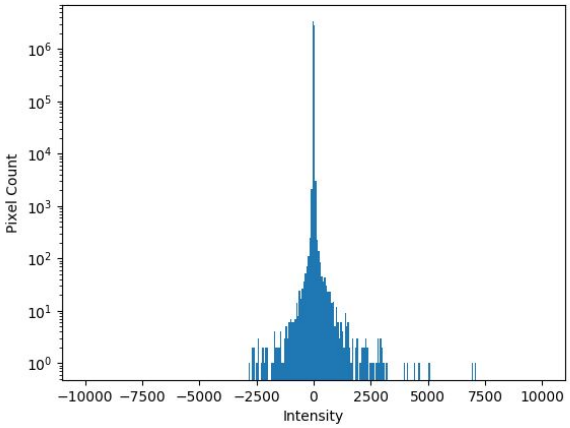

Figure 16. Histogram showing the number of pixels (pixel count) in the residual image from Figure 15 c with a given flux intensity. The mean of this histogram is -3.47, standard deviation is 22.01, and RMS is 3.48. Bin size is the calculated signal to noise ratio of the image. Compared to Figure 17, this histogram is markedly better in terms of symmetry and closeness to a Gaussian distribution.

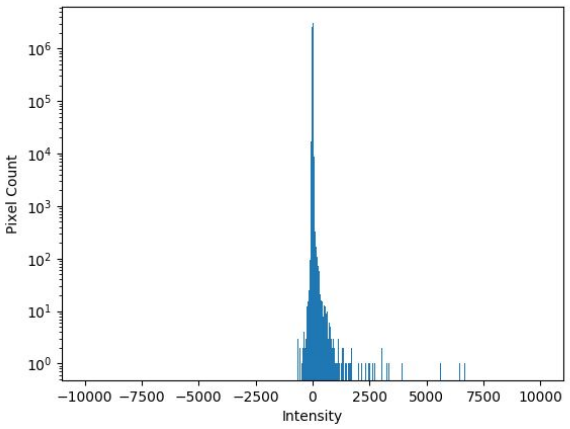

Figure 17. Histogram showing the number of pixels (pixel count) with a given flux intensity for the same image as Figure 16 but subtracted with non-adaptive Bramich. The mean of this histogram is 7.40, standard deviation is 24.34, and RMS is 7.40. Bin size is the calculated signal to noise ratio of the image. Compared to Figure 16, this histogram and associated residual is markedly worse. Notice the high degree of asymmetry in the distribution.

From each subtracted image, residual sources are extracted using Source Extractor Python (SEP). We use SEP rather than SExtractor because it works faster in a pipeline, and the data types are easier to input into our databases because it is a native python library. On linear detectors, the value measured at each pixel is the sum of a "background" signal and light coming from the sources of interest. To detect faint objects and make accurate measurements, the background is estimated before source detection. In SEP, source detection and background estimation are two separate steps (Barbary (2016)). First, the subtracted residuals are passed through the `sep.Background()` function, which carries out background estimation by providing a representation of spatially variable image background and noise.

Then, the background subtracted data residuals are passed through the `.extract()` function, which carries out source detection and extraction. The minimum threshold for pixel value is set to three times the mean of the image background in order to reduce SNR (Barbary (2016)). In the end, SEP creates a segmentation map, which returns the member pixels of each object, as well as a list of parameters associated with each source.

2.11. *Section Identification*

Since we have created an Andromeda mosaic (See figure 5) to overview and direct our Andromeda observations, this section identifier number is a useful HDU to attach to each section. Thus, our pipeline's option `SECID` step will input the HDUL and return the HDUL with an additional HDU found by comparing the RA and DEC information to the section bounds defined in our mosaic.

2.12. *Light Curve Production*

Alongside our efforts in automated transient detection through DIA, we are actively developing a detection system based on the automatic collection and analysis of light curves. This method allows us to evaluate transient candidates based on more detailed information about changes over time rather than on a single subtraction.

We first calculate the magnitude of each star. We convert the flux to magnitude by normalizing the fluxes with respect to reference stars using Gaia DR3 data (Kostrzewa-Rutkowska et al. (2018)). A set of reference stars across all images are selected by excluding potential transient candidates found by subtract and querying the Gaia DR3 database with the coordinates of the non-variable sources. The instrumental magnitude of both the reference and target stars are calculated using an aperture photometry function based on the `photutils` package. Source extractor calculates the position of the centroids, and the radial size of each source. Then `photutils` calculates the sum of pixels in that aperture. The flux is then calculated by multiplying the sum by the gain, which then gets converted to magnitudes.

The Gaia magnitudes for each of the reference stars are converted from the Gaia g filter to the filter in which

images were taken (generally SDSS g') using the prescribed methods in Kostrzewa-Rutkowska et al. (2018). Stars that are dimmer than 20th magnitude and stars that are not present in every image are removed from the reference star list. Then a linear fit between the instrumental magnitude and Gaia magnitude for the remaining reference stars is taken to find the zero point of the image. This zero point calibration is used to determine the magnitude of the target star in the accepted frame of the sky. This is the magnitude then utilized in constructing our light curves. The error in magnitude is derived from the calculated error in flux due to photon noise, background noise, and read noise in equations 10.

$$\delta M \approx \frac{1.086}{g^2} \frac{\sqrt{g(N_{\rm back}\pi(FWHM/2)^2 + 10^{-0.4M_{\rm inst}})}}{10^{-0.4M_{\rm inst}}} \tag{10}$$

Where $N_{\rm back}$ is an estimate of the background brightness of the image before subtraction, $FWHM$ is the full width half maximum of photutil's Gaussian kernel, g is camera gain, and $M_{\rm inst}$ is the instrumental magnitude of the source. After the magnitudes and errors are calculated, we use a Lomb-Scargle Periodogram algorithm provided by `astropy` to find the period of periodic sources, and finally phase fold using `PyAstronomy`'s phase folding routine.

Light curve analysis will be performed on more data obtained from our sources to identify and classify variables. The signal we are looking for is expected to be of a short time period with a larger magnitude than the background stars, so any such light curve should be easily distinguishable from variable stars. We do not plan to narrow our parameters for our specific search purposes, and will also analyze regular, and non-variable sources. We propose to use the method described in Moretti et al. (2018), as well as a method similar to the one employed in VaST (Sokolovsky & Lebedev (2018)).

3. RESULTS

3.1. *Preprocessing and processing speed*

Before the integration of multiprocessed image alignment and GPU accelerated image subtraction, these steps took an average of 11.6 and 21.1 seconds, respectively, for 12 Mpixel LCO images. This resulted in a time per image average of 35 seconds. After these two changes, our pipeline now averages below 3 seconds for 12 Mpixel images. Assuming linear scaling, we expect approximately 6.5 seconds to process a QHY image.

The time for preprocessing is currently 90 minutes per 1,000 images. This result of 5.4 seconds per images is longer than we desire. However, no efforts have been made to accelerate preprocessing at all. By simple multiprocessing, we expect to accelerate this process by a factor of 10 (with the theoretical limit being 72, our number of cores) as we saw for alignment. With these projected optimizations, we are on track to meet our criteria of 10 second image processing for Broida rooftop data.

3.2. *Subtraction Capability*

Using characteristic star fields simulated by the STUFF and SKYMAKER programs, We tested the lowest magnitude that we can consistently detect by using simulated images including transient sources through the STUFF and SKYMAKER programs. We used the STUFF program to create catalogs of stars and their respective magnitudes and then used SKYMAKER to generate a set images containing stars that aligned with the catalogues created by STUFF. In order to test different magnitudes, we generated images with and without transients and ran them through our pipeline. As seen in Figure 18, we found that our pipeline could detect up to magnitude 19 stars with about 85% consistency. These data strongly suggest that we would have a high probability of detecting a Class-IV civilization's directed energy source originating in Andromeda, which we estimate to have a magnitude of 16, as calculated in Section 1.

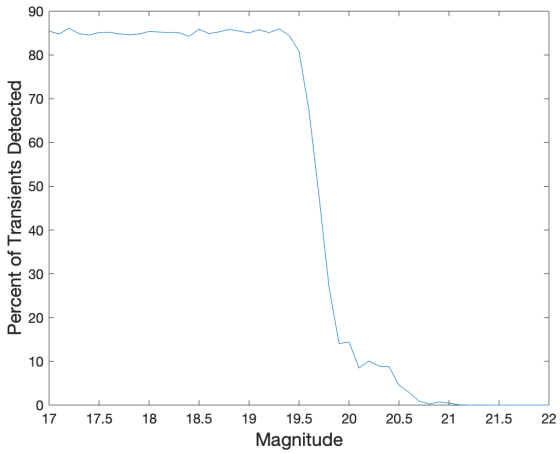

Figure 18. Graph showing the detection percent of transient sources in simulated images. Magnitudes were tested in intervals of .1 magnitude for magnitude 17 to 22 transients.

Next, we calculated a false positive rate for identifying transient sources this time using fully simulated images. To create the simulations, we used the Astromatic software STUFF and SKYMAKER (Bertin (2009)) for simulating full fields of stars, using similar methods as in Sánchez et al. (2019). Finally, we compared the extracted sources to the locations of any inputted tran-

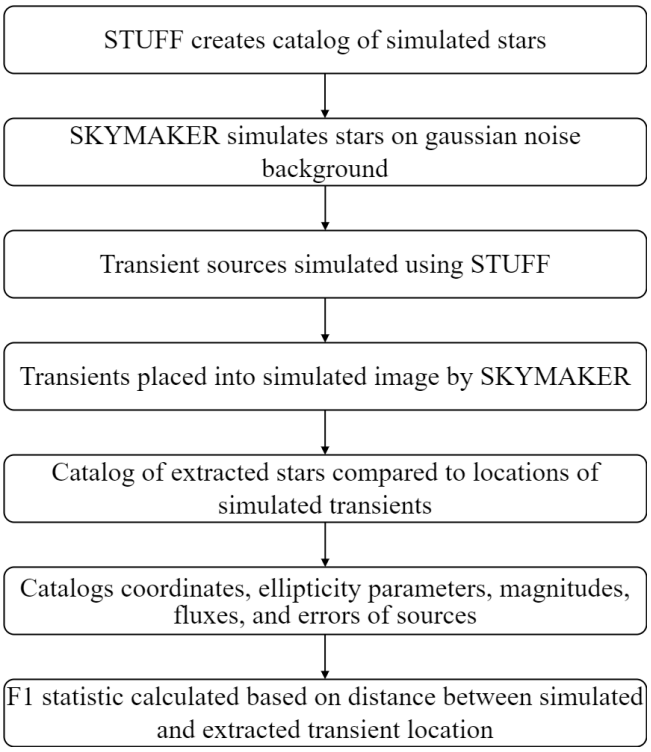

Figure 19. Flowchart that presents how we simulated our sets of 100 images for transient detection. We begin by creating our simulated images using STUFF to fabricate sources which are then placed on a Gaussian noise background by SKYMAKER with various S/N's. Transient sources are also created using STUFF and subsequently placed into one of the simulated images by SKYMAKER. These images are then run through our pipeline and the SEP catalog is compared with the locations of the inputted `sources`. From the discrepancies between these two lists, the F_1 statistic is calculated.

sients to determine a false positive rate for our pipeline. See Figure 19 for a detailed chart of our method.

We chose to use the F_1 statistic to measure the accuracy of our pipeline. To determine this statistic, we used the following formula:

$$F_1 = 2T_P/(2T_P + F_N + F_P)$$

where T_P is the number of true positives (sources correctly identified as transients), F_N is the number of false negatives (transient sources not detected by our pipeline), and F_P is the number of false positives (sources incorrectly identified as transients). As described in Sánchez et al. (2019), the F_1 statistic is extremely useful in weighing the effect between missing transients and incorrectly identifying non-transient objects. We found the DaLEC pipeline in simulated testing has an F_1 statistic of .38, which is close to values found in the Sanchez paper which found F_1 statistics between .37 and .55 in their testing.

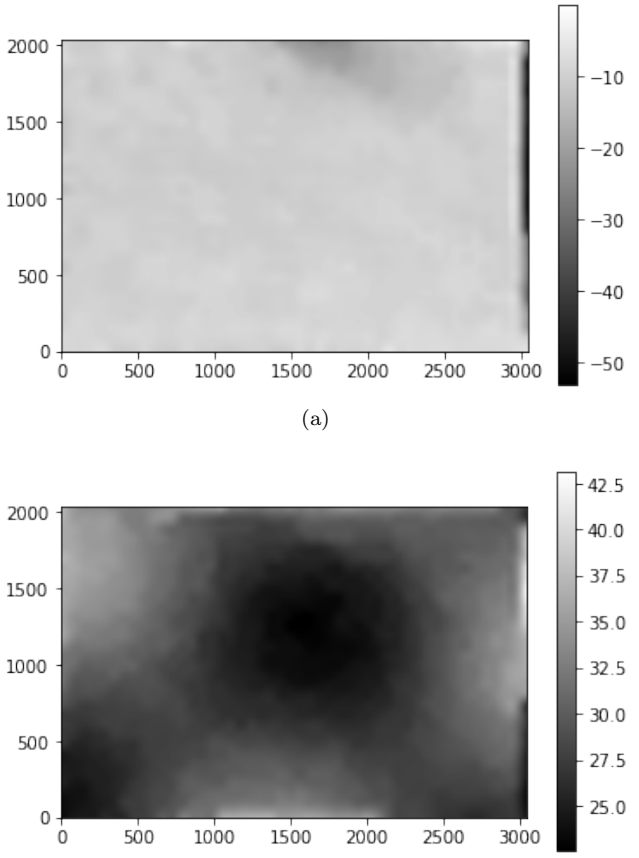

Figure 20. Images demonstrating the background data obtained using SEP on a image of section 24 of M31 run through the pipeline.(a) represents the `sep.Background().back`, the background noise of the image. (b) represents the RMS of the background of the image.

3.3. *Light Curve Results*

We have early light curves, but are not satisfied with the periodogram yet. 21 is the best light curve we have produced compared to PTF.

4. CONCLUSION

In summary, the DaLEC pipeline is a new difference image analysis pipeline and database for transient detection with small telescopes. The project currently focuses on analyzing Andromeda galaxy due to its relative proximity, large size, and brightness. Furthermore, it is an ideal candidate for the detection of directed energy sources.

We have been focusing on ensuring the pipeline detects sources which match up with known transients and plan on expanding to more methods of image analysis and data collection. These will give a better picture of what exactly the telescopes and pipeline are detecting.

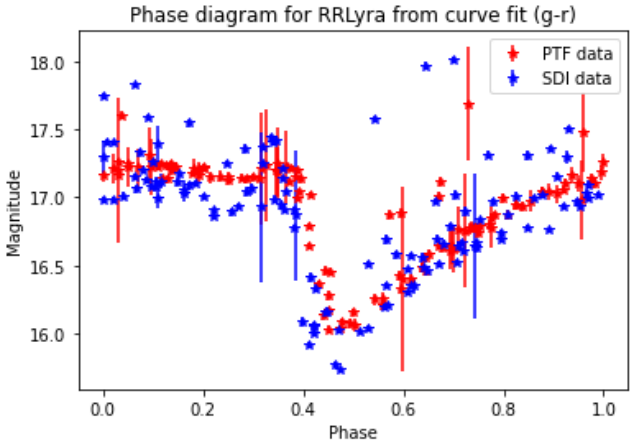

Figure 21. A light curve produced by photometry via `scipy.curve_fit` with errorbars corrected via PTF terms

4.1. *Future of the Project*

4.2. *Autonomous Data processing*

With the existence of command line integrated preprocessing and processing scripts, it is now possible to autonomously process data. By writing bash scripts for this task, we will automatically detect, preprocess, and process new data from our observatory. In order to use reliable templates, the image processing will trail the telescope by an observing session, beginning to process immediately after the last image is written and pausing after outputting processed data since it will be able to process all data before new data is available as desired.

4.2.1. *Machine Learning*

We are currently exploring the use of machine learning and neural networks in an effort to use newer methods obtain more accurate and streamlined transient detection.

In the future we plan on employing machine learning algorithms to identify and classify transients. To test our pipeline we processed images containing the GT And RR Lyrae variable star from the Intermediate Palomar Transient Factory **?**. This star was chosen as it lies within M31 and is bright enough to be easily detectable by our pipeline. We ran those images through the pipeline, constructed the light curves, and identified the sources using its RA and DEC coordinate. There were a total of 367 transient candidates picked up by the pipeline using the Bramich subtraction method. We reconstructed the phase diagram from the catalog file obtained from PTF's databases, and compared the phase diagram constructed from magnitudes found by our pipeline, and a close match was achieved, indicating that the pipeline was working as expected.

4.2.2. *Multi Dichroic Imaging System*

We are in the process of designing and building a multi dichroic imaging system to image in several different filters simultaneously. This allows us to maximize data collected per image and provides information about the wavelength of the incident light, aiding in transient classification.

ACKNOWLEDGEMENTS

We would like to thank the undergraduates who worked on this project before us. In addition, this work makes use of observations from the Las Cumbres Observatory global telescope network, and we are grateful for the support of the network and their generous allotments of observing time.

Funding for this project comes from the UCSB Faculty Research Assistance Program, NASA grants: NIAC Phase I DEEP-IN – 2015 NNX15AL91G and NASA NIAC Phase II DEIS – 2016 NNX16AL32G, the NASA California Space Grant (NASA NNX10AT93H), as well as a generous gift from the Emmett and Gladys W. fund.

REFERENCES

Barbary, K. 2016, Journal of Open Source Software, 1, 58, doi: 10.21105/joss.00058

Beroiz, M., Cabral, J. B., & Sanchez, B. 2020, Astronomy and Computing, 32, 100384, doi: 10.1016/j.ascom.2020.100384

Beroiz, M., Sánchez, B., & Iyer, V. 2020, toros-astro/ois: Version 0.2, v0.2, Zenodo, doi: 10.5281/zenodo.4042147

Bertin, E. 2009

—. 2017. https://readthedocs.org/projects/sextractor/downloads/pdf/latest/

Brown, T. M., Baliber, N., Bianco, F. B., et al. 2013, PASP, 125, 1031, doi: 10.1086/673168

Drake, A. J., Djorgovski, S. G., Mahabal, A., et al. 2009, Astrophysical Journal, 696, 870, doi: 10.1088/0004-637X/696/1/870

Harris, C. R., Millman, K. J., van der Walt, S. J., et al. 2020, Nature, 585, 357, doi: 10.1038/s41586-020-2649-2

Hu, L., Wang, L., Chen, X., & Yang, J. 2022, The Astrophysical Journal, 936, 157, doi: 10.3847/1538-4357/ac7394

Kostrzewa-Rutkowska, Z., Jonker, P. G., Hodgkin, S. T., et al. 2018, Monthly Notices of the Royal Astronomical Society, 481, 307, doi: 10.1093/mnras/sty2221

Lubin, P. 2016, Reviews in Human Space Exploration, 58, doi: 10.1016/j.reach.2016.05.003

McConnachie, A., Irwin, M., Ferguson, A., et al. 2004, Monthly Notices of the Royal Astronomical Society, 356, doi: 10.1111/j.1365-2966.2004.08514.x/abs/

McCully, C., Turner, M., Collom, D., & Daily, M. 2019, BANZAI Documentation. https://banzai.readthedocs.io/en/latest/

McCully, C., Turner, M., Volgenau, N., et al. 2018, LCOGT/banzai: Initial Release, 0.9.4, Zenodo, doi: 10.5281/zenodo.1257560

Miller, J. P., Pennypacker, C. R., & White, G. L. 2008, PASP, 120, 449, doi: 10.1086/588258

Moretti, M. I., Hatzidimitriou, D., Karampelas, A., et al. 2018, Monthly Notices of the Royal Astronomical Society, 477, 2664, doi: 10.1093/mnras/sty758

Ronacher, A. 2022, Click Documentation. https://click.palletsprojects.com/en/8.1.x/#documentation

Sánchez, B., Domínguez R., M. J., Lares, M., et al. 2019, Astronomy and Computing, 28, doi: 10.1016/j.ascom.2019.05.002

Sokolovsky, K., & Lebedev, A. 2018, Astronomy and Computing, 22, 28–47, doi: 10.1016/j.ascom.2017.12.001

Stewart, A., & Lubin, P. 2017, SPIE, 61, doi: 10.1117/12.2286945

5. APPENDIX: PYTHON CODE

5.1. *setup*

`setup.py` enables the pipeline to be installed via pip3 so it can be evoked from the command line. The setup script installs all necessary packages optimized for a minimal size virtual environment.

```
from setuptools import setup, find_packages
import subprocess
import sys

'''
try:
    import numpy
except ModuleNotFoundError:
    import sys
    sys.exit("numpy not found, pits requires numpy for installation.\n Please try '$pip3 install
                                                    numpy'.")
try:
    import setuptools_rust
except ModuleNotFoundError:
    import sys
    sys.exit("setuptools_rust not found, pits requires setuptools_rust for installation.\n Please
                                                    try '$pip3 install setuptools_rust'.")
'''

subprocess.run(sys.executable + " -m pip install --upgrade pip setuptools==56.0.0 setuptools_rust
                                                numpy", shell=True)

setup(
    name="pits-cli",
    version="0.99",
    py_modules=["pits"],
    # packages=find_packages(include=["openfits"]),
    include_package_data=True,
    install_requires=["click", "astropy", "astroalign"],
    entry_points="""
        [console_scripts]
        pits=pits._cli:cli
    """, #Names the pipeline pits and invokes the cli command in pits._cli
)
```

5.2. *__init__*

`__init__.py` initiates the pipeline by pointing click to all modules that are part of the pipeline. Modules not in the script are not passed to the pipeline.

```
import os

from .fitsio import read
from .alignment import alignment
from .calibrate import calibrate
#from .stack import stack
#from . import test

from .fitsio import read_cmd as _read_cmd
from .alignment import alignment_cmd as _align_cmd
from .calibrate import calibrate_cmd as _calibrate_cmd
#from .stack import stack_cmd as _stack_cmd
```

5.3. *Alignment script with click*

```
import click
import os
import glob
import astroalign
import numpy as np
```

```
import time
# import re
from astropy.io import fits
from . import _cli as cli

timestart = time.time()


def alignment(hduls, name):
    try:
        data = hduls[name].data
    except KeyError:
        data = hduls["PRIMARY"].data

    reference = np.nan_to_num(hduls[0][0].data[:-35, 40:-4], copy=True, nan=0.0, posinf=None, neginf
                                                        =None)

    for i, hdul in enumerate(hduls):
        #with fits.open(path) as hdul:
        data = data[:-35, 40:-4]
        timestart1 = time.time()
        try:
            print('Byte order not swapped.')
            output = astroalign.register(data, reference)[0]
            hdul['PRIMARY'].data = output  # writing aligned data to hdu
            print(f'frame {i} aligned in {time.time()-timestart1}')

        except (ValueError):  # Catches byteorder error
            print('Byte order swapped.')
            output = astroalign.register(
                hdul['PRIMARY'].data.byteswap().newbyteorder(),
                reference, min_area=10)[0]
            hdul['PRIMARY'].data = output  # writing aligned data to hdu
            print(f'frame {i} aligned in {time.time()-timestart1}')
        print('Aligned in :', time.time()-timestart, ' seconds\n')
    return hduls

@cli.cli.command("alignment")
@click.option("-n", "--name", default="SCI", help="The HDUL to be aligned")
@cli.operator

def alignment_cmd(hduls, name="SCI"):
    """
    rotates frames and preprocessing image alignment
    """
    return alignment([hduls for hduls in hduls], name)
```

5.4. *Calibrate Script with click*

```
import click
import os
import glob
import numpy as np
import time
from astropy.io import fits
from . import _cli as cli

# Function returns master bias frames for correcting images.


def bias(bias_path, output_path=None, read_ext='Primary'):
    timestart = time.time()

    bias_images = []
    for i, img in enumerate(bias_path):
        with fits.open(img) as frame:
            bias_images.append(frame[read_ext].data)
```

```
        print('bias: ',i)
    main_bias_data = np.median(np.array(bias_images), axis=0)
    print('master bias created in :', time.time()-timestart, ' seconds\n')
    main_bias = fits.PrimaryHDU(data=main_bias_data)
    if output_path is not None:
        MBpath = os.path.join(output_path,'main_bias')
        os.makedirs(MBpath)
        main_bias.writeto(MBpath, overwrite=True)
    return main_bias

def dark(dark_path, output_path=None, read_ext='Primary'):
    timestart = time.time()

    dark_images = []
    for i, img in enumerate(bias_path):
        with fits.open(img) as frame:
            exptime = frame[0].header['EXPTIME']
            dark.append(
                np.divide(
                    np.subtract(
                        frame[read_ext].data,
                        bias.data),
                    exptime)
                )
        print('dark: ',i)
    print('Combining...')
    dark = np.median(np.array(dark), axis=0)
    main_dark = fits.PrimaryHDU(data=dark)
    print('master Dark created in :', time.time()-timestart, ' seconds\n')
    if output_path is not None:
        MDpath = os.path.join(output_path,'main_dark')
        os.makedirs(MDpath)
        main_bias.writeto(MDpath, overwrite=True)
    return main_dark

def flat(bias_images, dark_images, output_path=None):
    timestart = time.time()

    flat = []
    for i, img in enumerate(flatimages):
        with fits.open(img) as frame:
            exptime = frame[0].header['EXPTIME']
            # double subtract here
            flat.append(np.subtract(np.subtract(
                    frame['Primary'].data, (dark.data * exptime)),
                    bias.data))
        flat[i] /= np.median(flat[i])  # check this step
        print('flat: ',i)
    flat = np.median(np.array(flat), axis=0)

    main_flat = fits.PrimaryHDU(data=flat)
    print('master Flat created in :', time.time()-timestart, ' seconds\n')
    if output_path is not None:
        MFpath = os.path.join(output_path,'main_flat')
        os.makedirs(MFpath)
        main_bias.writeto(MFpath, overwrite=True)
    return main_flat

# does a function need to return anything? I could return all hduls in a list.
def calibrate(hduls, bias_images, bias_path, dark_images, dark_path, output_path, read_ext):
    """
    Parameters
    ----------
    write_to_disk : Bolean
    Option to write output to disk. If true, writes to specified location.
    Returns
    -------
    hduls_list_CAL : List
```

```
    A list of hdul's with the calibrated fits data writen to the primary hdu

    """
    timestart = time.time()

    # Ensure that files are sorted by the index in file name.
    sciimages = sorted(sciimages)
    # Calls main bais functions and unpacks tupple return
    main_bias = bias(bias_images, output_path, read_ext)
    main_dark = dark(dark_images, output_path, read_ext)
    main_flat = flat(bias_images, dark_images, output_path)

    for i, img in enumerate(sciimages):
        with fits.open(img) as hdul:
            print('calibrating: ',i)
            exptime = hdul[read_ext].header['EXPTIME']
            bias_apl = np.subtract(hdul[read_ext].data, main_bias.data)
            bias_dark_apl = np.subtract(bias_apl, main_dark.data * exptime)
            sci = np.divide(bias_dark_apl, masterflat.data)
            hdul[read_ext].data = sci
            if output_path is not None:
                Calpath = os.path.join(output_path,'Cal')
                os.makedirs(Calpath)
                main_bias.writeto(Calpath, overwrite=True)
    print('Images calibrated in:', time.time() - timestart, ' seconds\n')
    return hduls_list_CAL  # Returns list of hdul's

@cli.cli.command("calibrate")
@click.option('-b', '--bias_path', type=str, help="Specify path to directory of fitsfiles.",
                                                    required=False)
@click.option('-d', '--dark_path', type=str, help="Specify path to directory of fitsfiles.",
                                                    required=False)
@click.option('-w', '--output_path', type=str, help="Specify path to directory to save preprocessing
                                                     fitsfiles.", default="./")
@click.option('-r', '--read_ext', default='Primary', help="The HDU extension to be aligned.")
@cli.operator

def calibrate_cmd(hduls, bias_path, dark_path, output_path=None, read_ext='Primary'):
    """
    calibrates the image with bias images and flatfielding
    """
    if bias_path is None:
        try:
            bias_path = askdirectory()
        except:
            click.echo("Visual file dialog does not exist, please use option -d and specify path to
                                                                directory to read fitsfiles.", err=
                                                                True)
            sys.exit()
    if dark_path is None:
        try:
            dark_path = askdirectory()
        except:
            click.echo("Visual file dialog does not exist, please use option -d and specify path to
                                                                directory to read fitsfiles.", err=
                                                                True)
            sys.exit()
    hduls = calibrate(hduls, bias_path, dark_path, output_path=None, read_ext='Primary')
    if hduls:
        return hduls
    else:
        sys.exit(f"Could not open fitsfiles from directory {directory}")
```