UNIVERSITY OF COPENHAGEN

A thesis presented to the Faculty of Sciences in partial fulfillment of the requirements for the degree

Master of Science in Physics

Fast transient detection

Master Thesis

Flemming Daniel Fischer

rdl821@alumni.ku.dk

Supervisor

Johan Peter Uldall Fynbo

jfynbo@nbi.ku.dk

August 15th, 2023

Pages: 75

Nr. of characters: 163414

ABSTRACT

Gamma-ray bursts are transient events and some of the most luminous events known. A gamma-ray burst consists of a prompt emission that only lasts minutes and a longer fading afterglow. As the prompt emission is gone after a few minutes after the event occurs, fast and robust tools are important in the pursuit to get useful data on the prompt emissions of gamma-ray burst.

In this project, the tool PyNOT will be tested thoroughly for the purpose of identifying gamma-ray burst events at their early stages at the Nordic Optical Telescope at La Palma. PyNOT will be tested on imaging data from different gamma-ray burst events with the purpose of exposing PyNOTs strengths and weaknesses. Further, this project will investigate, develop and test methods not already exploited by PyNOT with the intention of implementing them in PyNOT for even more robust transient identification in images. Moreover, it will be bared in mind that the Nordic Optical Transient Explorer in the near future will be up and running. Thus, making PyNOT more versatile for other instruments is investigated as well.

A conclusion of PyNOTs current capabilities will be made as well as recommendations for further development on PyNOT with the intention to use PyNOT for fast transient detection on the Nordic Optical Telescope will be elaborated.

ACKNOWLEDGEMENTS

First, I will thank my supervisor Johan Peter Uldall Fynbo. A huge thank you for giving me the opportunity to work on this project. And thank you for the good and inspirational discussions as well as the good inputs and ideas you have provided me with throughout this whole project.

Moreover, I will thank Jens Krogager for helping me resolve the initial problems PyNOT had when I started this project.

And lastly, I will thank Sofie Pedersen for supporting me and helping me by reading through my report and giving inputs for improvements.

CONTENTS

PROBLEM STATEMENT

Transients are some of the most difficult objects when it comes to making good observations because the time between detection and observation is critical. The sky is not constant but it is in constant change as transient sources are turning on and off on a wide range of timescales. Those known with shortest timescales like fast radio bursts and gamma-ray burst are especially difficult as they fade within milliseconds and minutes, respectively. For those events a significant amount of information is lost do to the slow manual process we use today for detection of transients. Therefore, the information obtained from those events is mainly from the afterglows. For this reason automatic detection of very fast transient events are very important to get good quality data.

In the thesis I will try to test already existing software for this exact purpose. I will explore what works and what is in need of improvement. Concretely, I will test the PyNOT module, AutoPhotometer and STDPipe and do some statistics to argument for their robustness. I will also try to come up with ideas for further development and develop implementations of some of these.

1 INTRODUCTION

1.1 A brief history of transient events

1.1.1 *Transient events*

Transients are a broader term for short-lived events with a time span that vary from a few seconds to years. Some events may last years and it may sound like a long time, but this is a brief moment on a cosmological scale. Transient events occur frequently in our universe and we know that they have been known for centuries as the first supernova was recorded in 185 AD by Chinese astronomers. The transient event they recorded is today known as a supernova named SN185 which lasted for months [1]. Since then, several transient events have been recorded in the following centuries. However, it is mostly longer lasting transient events such as supernovae that have been observed. The shorter transient events like Gamma Ray Bursts(GRB) were first detected in 1967 by the *Vela* satellites [2]. Those satellites were sent into orbit to monitor gamma radiation due to the threat of nuclear bombs at the time. The Vela mission was initiated as a result of the Limited Test Ban Treaty made in 1963 between the USA and the Soviet Union. The treaty banned the testing of nuclear weapons in the atmosphere, outer space, and under water [3]. However, in 1967 the Vela satellites received short lasting bursts of gamma rays from what seemed to be arbitrary directions. The gamma rays directions did not seem to come from any particular source meaning they must have been coming from outer space. Those results were first published years later in 1973 due to the mission being classified, making this year the beginning of the era of GRBs[4].

Since then, more satellites have been launched with the purpose of gathering information on GRBs. Until 1997, observations of GRBs had been made of the afterglows of the GRBs that can last for days, and not of the prompt emission. The lack of observations of the prompt emission until 1997 is due to the prompt emission of GRBs being especially short-lived as it is only lasting up to minutes. Nevertheless, in 1997, GRB9903271 was detected and observations began already 22 seconds after. This is one of the fastest observations made of a GRB event and resulted in data gathered on the prompt emission. This fast observation was possibly due to the fast and precise coordinates determined for the event. The observation greatly improved the understanding of the origin of the GRBs[5]. Despite this fast and good observation, most observations today are still taken of the afterglows and not the prompt emission due to the time that passes from the first detection to locating and swiveling telescopes to be pointing at the event. Because of this, it has for the recent decays had growing importance in automating the detection of GRBs, as the prompt emission is still very difficult to observe. This makes some of the current knowledge in the field of origin of the GRBs prompt emission lacking due to this problem. [6].

1.2 Current Gamma-ray burst theory

GRBs are some of the most energetic events with energy releases of $10^{48} - 10^{52}$ erg, making them hundreds of times more energetic than a typical supernova. This intense release of energy in such a short amount of time puts GRBs among the most violent and luminous events known[7] [8]. The spectral distribution of gamma-ray bursts is peaking in the gamma-ray band, hence also the name[9]. The energy output from a gamma-ray burst can in the short moment they occur, outshine all other gamma-emitting sources in the universe. [2]. A GRB event can be divided into two parts, namely the prompt emission and the afterglow. However, before diving into what prompt emissions and afterglows are and the underlying physics, it makes sense to first understand what happens up to a GRB in the first place.

Causes of GRB events can also be classified into two groups, namely short GRBs and long GRBs. GRBs lasting less than two seconds are classified as short GRBs, while those lasting longer, some up to minutes, are classified as long GRBs.

The longer bursts are the most common type of GRBs. This type of burst is associated with the death of massive rotating stars with masses larger than 15 solar masses[8]. When massive stars are in the end stage of their life cycle, they begin to fuse iron in their core. This fusion process requires more energy than it is producing. This negative energy production is inevitably leading to a core collapse as the outgoing pressure in the core no longer can sustain the pressure of gravity pushing inward. The result of this core collapse is that the star is ending its life as a supernova. A supernova is a violent event that is leaving a white dwarf or black hole as a remnant. Whether the remnant is a white dwarf or a black hole depends on the initial mass of the star. Many of those longer GRBs are more specifically associated with the supernova type Ic(SN Ic)[7]. As GRBs are associated with rotating stars has an important implication as it leads to the remnant spinning incredibly fast due to the conservation of angular momentum. This is a consequence of the remnant being a result of huge amounts of the star has been compressed into a much smaller and more dense object due to the gravitational collapse.

Whereas the long GRBs are associated with supernovae, the short GRBs are suspected for being related to the merger of two compact objects, like two neutron stars in a binary system. The merge of two compact objects leads to a fast spinning black hole. Even though it is two different progenitors with different timescales of the event occurring, both are leading to very similar circumstances. Namely a very compact object, often a black hole, with a temporally accretion disk shaped as a torus surrounding it. In the case of supernovae, the accretion disk tends to be more massive than the one formed by mergers[10]. However, what happens during the event is best explained by the fireball model, also often accepted as the standard model for GRBs.

1.2.1 The fireball model

As the remnant is a very compact object and the angular momentum is conserved, the remnant is spinning incredibly fast. This fast spin induces extremely strong magnetic fields along the axes of the rotation of the remnant. Charged particles surrounding the remnant are captured in those magnetic fields and get accelerated to relativistic velocities. The captured particles are then sent out along the axis of rotation where they transfer energy to photons. If the axis occurs to point in the direction of us, we will be able to see those photons[11].

When the matter is accelerated along the axis, it is expanding adiabatically out in a fireball through the surrounding matter. Due to the rapid expansion of the fireball an optically thick front is formed of the expanding matter. Due to the optical thick front, thermal radiation can not occur. Instead, radiation is done by dissipation of kinetic energy to the surrounding matter and photons through a mix of synchrotron radiation and inverse compton scattering. The accelerated particles are powered by the central engine which gets its energy from a combination of the free energy from the acreating matter from the torus falling in on the remnant and the angular momentum of the spinning remnant [12].

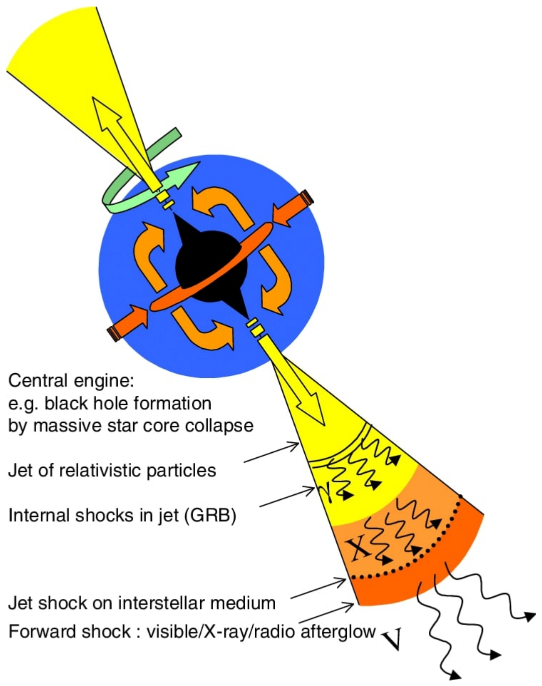

Central engine:
e.g. black hole formation
by massive star core collapse
Jet of relativistic particles
Internal shocks in jet (GRB)
Jet shock on interstellar medium
Forward shock : visible/X-ray/radio afterglow
γ
X
V

Figure 1: *In this figure we see an illustration of the event occurring in a GRB described by the fireball model. We see the central engine with its torus surrounding it and the acreating matter falling in. The matter is ejected along the axis of rotation and internal shocks happen due to relativistic charged particles catching up with slower moving shells further out. This makes internal shocks occur that generate gamma rays. Further out, we see that the jets collide with the external medium generating photons across many wavelengths [7]*

When the fireball expands, it creates both internal and external shocks. The matter ejected along the axis is not ejected in one shell, but instead in several shells. The shells are moving with varying Lorentz factors around 100, making the shells move with different relativistic velocities. The internal shocks happen in between the ejected matter itself when some of the ejected matter catches up with other ejected matter moving slower in a shell further out in the jet. This causes shocks to occur and strong magnetic fields are created in those shocks. These magnetic fields deflect some of the charged particles which leads to them emitting high-energy photons in the process called synchrotron radiation. The synchrotron radiation is believed to be the main reason for the prompt emission. However, it is also speculated that inverse compton scattering is also a contributor to the high energy photons emitted due to the highly relativistic electrons transferring energy to photons. Those shells stack up in a thin opaque wavefront, that only emits the radiation when the front is becoming optically thin, which is the main reason the prompt emission is so short-lived [12]. When inspecting spectra of GRB prompt emission, it is seen that they all vary, and many have a seemingly random number of spikes (Figure 2). Those spikes are believed to be accounted for by the internal shocks[12][4].

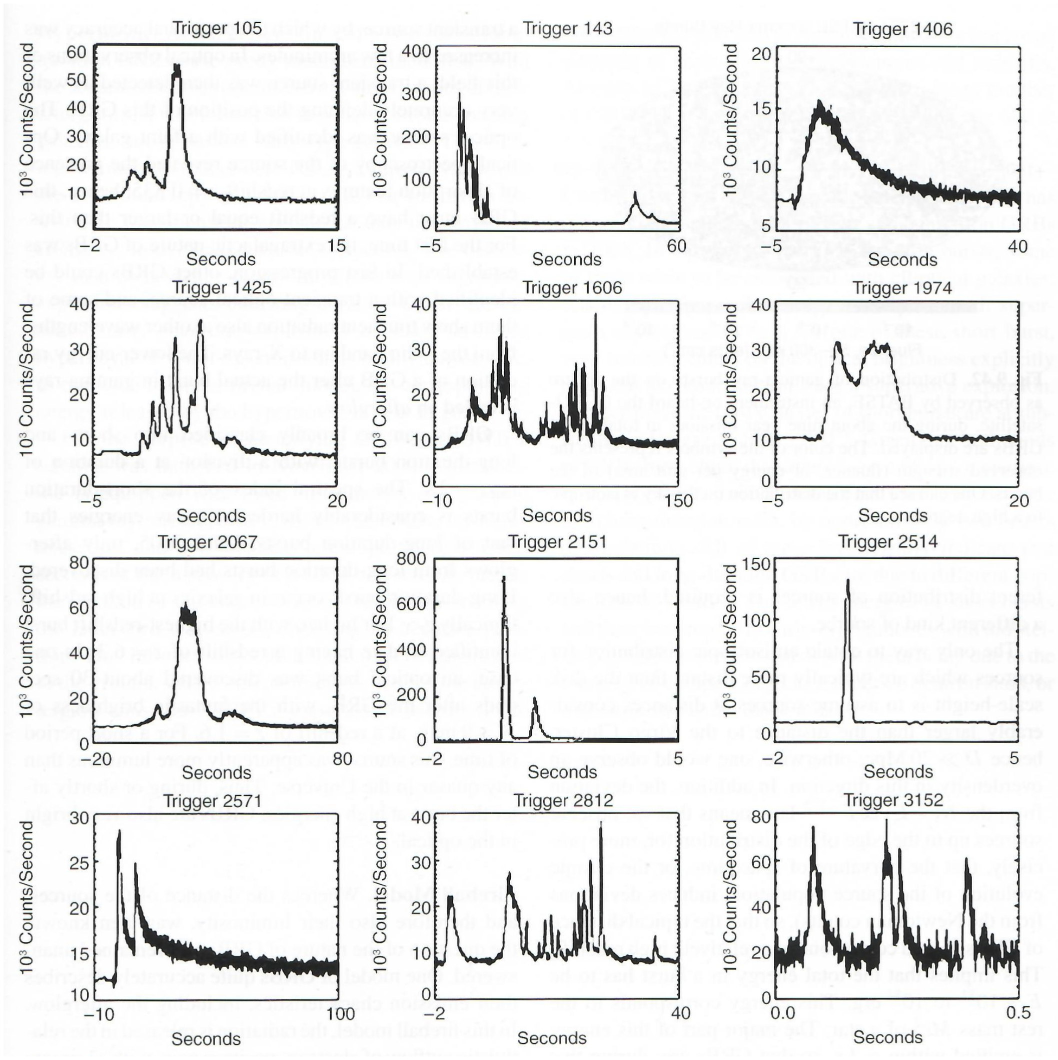


Figure 2: *This figure shows the prompt emission light curves of 12 different GRB events. The figure also shows that there are peaks in the GRB prompt emission and that those spikes differ from event to event. Those spikes are believed to be a result of internal shocks in the GRB described by the fireball model[4] [12]*

The external shocks occur when the jets interact with the surrounding matter of the remnant in the interstellar medium, making those shocks highly relativistic. It is those external shocks that create the long-lasting afterglow as the shock wavefront is interacting with the surrounding matter, again emitting synchrotron radiation. However, in this interaction with the surrounding matter, some of the radiation will also be thermal. At this moment, the thermal radiation has no problems escaping as the wavefront no longer is optically thick. The afterglow is much longer lasting as the jets are moving through a long distance of matter slowly cooling it down and no opaque wavefront is present[12].

In the time following a GRB, the afterglow is decaying as seen in figure 3. The magnitude of the early afterglows often has steady brightness before it breaks into a power law. As seen in figure 3, sometimes there even is an increase in magnitude before the break[13].

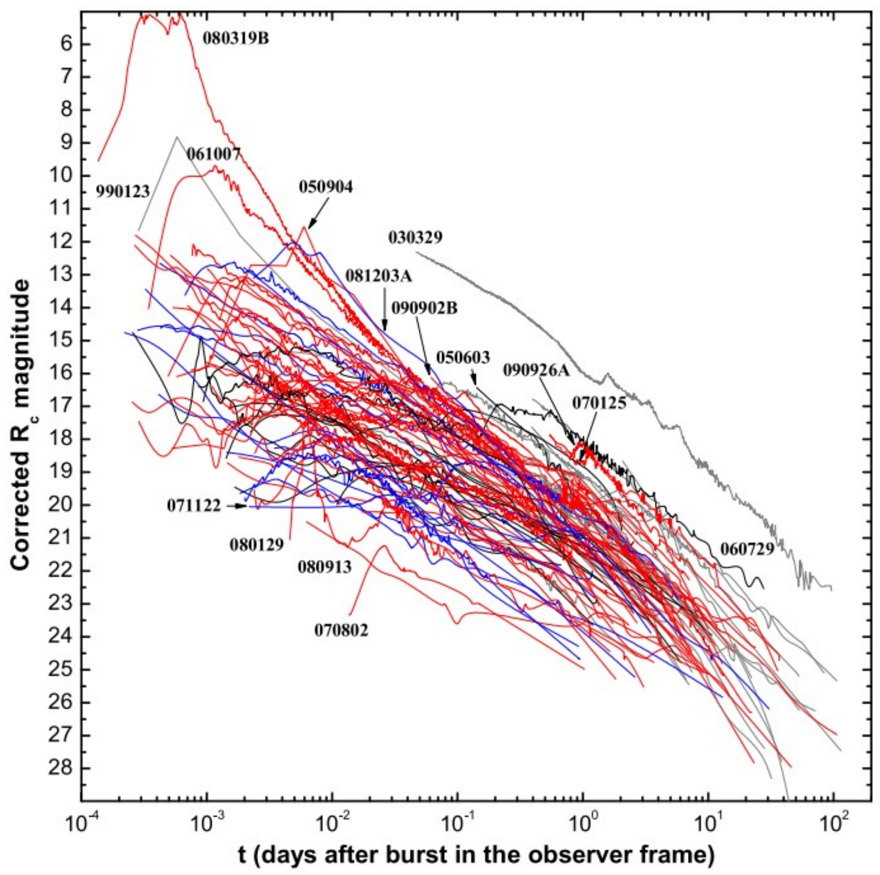


Figure 3: *Compilation of afterglows of early detected GRBs. It is seen that the early afterglows have a short steady time with little change in brightness. This is followed by a break. The afterglows of the GRB are following a power law after they break[13].*

1.3 The SWIFT satellite

The SWIFT satellite was launched in 2004 as part of The Swift Gamma-Ray Burst Mission with the purpose of exploring GRBs. The SWIFT satellite has three instruments onboard. One of the three instruments is the Burst Alert Telescope(BAT) which has a wide field of view of 100x60 degrees that measures light in the energy range of 15-150KeV. The BAT is an autonomous trigger system which only gets triggered by point sources like a GRB event. The BAT is very sensitive, making it trigger even weak differences in photon counting compared to the background. The BAT also locates the trigger source with an error of one to four arcminutes and distributes those results to the Gamma-ray Coordinates Network(GCN) [14]. The GCN is a network for scientists where coordinates of GRBs are distributed. Further BAT uses the detected position to swivel its two other instruments, the X-ray telescope(XRT) and the Ultro-violet/Optical telescope(UVOT), in the direction of the GRB event that triggered the BAT[15].

The XRT and UVOT instrument can swivel in the direction of the GRB in typically less than 100 seconds. The XRT instrument can be used to estimate the location of the GRB within only five to six arcseconds and distribute these more precise coordinates to the GCN. With ground analysis of the XRT coordinates, the position can be estimated down to two to three arcseconds. However, with ground analysis, the coordinates will take even longer before being distributed. Nevertheless, if estimated with ground analysis, these coordinates are then also distributed to the GCN. However, without ground analysis, SWIFT can provide very accurate positioning with XRT of the GRB typically within a time period of 100 seconds. The UVOT can then take a spectrum and make photometry in the optical wavelengths from 170-600nm[15]. The autonomous detector BAT and the fast swivel of XRT have huge implications for fast transient detection as they can provide quite accurate positions of GRBs to the GCN very fast.

1.4 Introduction to this work

When receiving a signal from a GRB, it is crucial finding its precise origin as fast as it will otherwise quickly have faded and the earliest stages of the GRBs are lost. It is those early stages that are of special interest as it is here the data is lacking. At the moment, when detecting a signal with the SWIFT satellite, the coordinates have to be gathered through the GCN. The more accurate and precise location of the GRB has to be done manually here on Earth by inspecting images taken with ground-based telescopes, including the Nordic Optical Telescope(NOT). This is done in the middle of the night by someone who might be tired and have to spend time going through catalogs of the night sky and comparing those catalogs with objects in the image taken by the ground telescope. This could be done much more effectively as a computer should be able to handle the exact same process. If the program is built sufficiently well, it could maybe be even more robust in giving the exact location. The biggest advantage of a computer is its speed. With a computer, it should be possible locating a transient in an image almost immediately. For that reason, a fast and automatic pipeline could result in a very fast observation of a GRB. Hence, an automatic pipeline for the detection of GRBs could lead to more observations of the prompt emissions and not just the afterglows. Therefore, a pipeline could be a good solution to replace the current process of identifying GRBs.

Already existing tools like PyNOT are partly doing this. PyNOT is a tool in Python that has automated the process of processing the images for easy comparison of a catalog and it even has a function that finds transient. However, even PyNOT is not fully automated as you have to help it a little with some inputs. For the pipeline to be as optimal as possible it should not only be fast but also very easy to use, just as it also has to be very robust in its detection.

By testing PyNOT and other existing tools, I will explore what methods for fast transient detection that work and what methods need to be improved. Further, I will be able to give a recommendation on the current best tool for transient detection for NOT. Moreover, the test can provide valuable information about what features a tool might need to be robust enough to be implemented in a fully automated pipeline in the future.

1.5 Methods for finding transients

In the search for transients, telescopes like the BAT and the Gamma-ray Burst Monitor(GBM) on SWIFT and Fermi respectively are constantly, among several other telescopes, searching large areas of the sky with a wide field of view for unexpected bursts of energies[16]. The telescopes then autonomously distribute the coordinates and the area within the burst was detected through the GCN. This way the GCN has the ability to share information about the burst, while the burst is still ongoing, to everyone interested in transient research, whether it is professionals or amateur astronomers[15][14].

In general, when trying to find transients in the sky there are different main methods that could be used. In this report, I will exclusively focus on methods used to identify a transient source in an image. There are two main methods that I will be looking into. One of them is using known properties of the event to try to detect the transient. For a GRB, we know the fade rather quickly. So one method is looking for fading objects. However, this requires several photos with time in between and each photo itself takes time, which results in it being a poor method for finding transients fast.

Another approach is to exclude as many other candidates in an image as possible or rank them depending on conformities with expectations about a transient event. As a GRB is a brief event that happens as we observe, we do not expect to see them appear in a catalog of stellar objects. For that reason, we could lower the priority that should be given to the sources in an image that are already in a catalog. Further, we know a GRB is a point source and it should however also lower the priority if a candidate is found not to be a point source. Moreover, it could be useful to compare with the uncertainties that come with the coordinates provided by the satellite that detected the transient. If a candidate is not consistent with the errors of the coordinates, it should also have a lowered priority. These methods are not dependent on several images being taken. Therefore, the methods mentioned are better suited for finding transients very fast. However, all of the above methods can be used for the detection of GRBs. For that reason, all the above methods will be reviewed in this report.

1.5.1 PyNOT

1.5.1.1 *Abilities of PyNOT*

One of the tools for transient detection is PyNOT. PyNOT is a library in Python that has several functions. It is a data processing pipeline made for the Alhambra Faint Object Spectrograph and Camera(ALFOSC) at the Nordic Optic Telescope which does data processing of long-slit spectroscopic data and imaging. For this project, we are only going to make use of the part of the PyNOT pipeline that does imaging. The pipeline consists of a series of tasks best run from the terminal. It does all the steps from classifying the data files to creating a corrected reduced image. This includes correction for cosmic rays, BIAS, and FLAT field correction and also subtracting the sky background. Besides that, PyNOT also handles world coordinate system(WCS) calibration by comparing it with the Gaia catalog. All of which, otherwise would have had to be done manually which is a long and time demanding process. Besides this, the pipeline is also capable of detecting sources in the reduced image and extracting them for later use. [17]

By comparing the sources detected in the image to those in the Gaia catalog the pipeline is supposed to be able to tell us which object is a new object and also where in the image the new object is located. This is a very important step for automatic detection of transient objects. Moreover, PyNOT displays error circles that correspond to the error for the BAT and XRT coordinates from SWIFT. PyNOT also provides information about whether the transient candidates are consistent with this error circle or not.

1.5.1.2 *Usage of PyNOT*

PyNOT is automated in such a way that one line of code executes a whole task. This way it is easy to do several steps fast with few lines of code. For example will the command "*init*" classify the raw data files and save the information in a file. In this file, it can be seen which files are raw images and which are BIAS or FLAT files. The pipeline also has other tasks that do several things at a time like the "*phot*" command. The command "*phot*" uses the raw data to create a reduced image and detect sources as well as estimate their magnitudes. Together with the "*findnew*" command that compares the found sources with the Gaia catalog, we have gone from raw data into classifying transient candidates in the sky with only three commands.

These three lines of code seem like all that is needed for fast and automatic detection of transient sources. However, the functions do need to get some good values for PyNOT to be able to find those transients robustly. And how robust it is has not yet been tested throughout. For that reason, more work and tests have to be done for PyNOT if it should be implemented and directly used together with ALFOSC.

2

THEORY & METHODOLOGY

2.1 World Coordinate System

In astronomy, the use of celestial coordinate systems is very important. Celestial coordinate systems are very similar to the geographical coordinate system used to specify a location on Earth. In the geographical coordinates system, the location of an object is often referenced with the use of the latitude and longitude of a sphere projected on the surface of the earth. The same system is used for celestial objects. However, instead of using latitude and longitude, the coordinates are most often referred to as right ascension(RA) and declination(DEC). In special cases, like for the galactic coordinate system, galactic height is used instead of declination. Sometimes it can also be referred to as galactic latitude(l) and galactic longitude(b)[18]. Nonetheless, the projected sphere is called the celestial sphere. The celestial sphere is separated into two hemispheres, divided by a celestial plane just like the geographical hemisphere is separated by the equator. Whereas the geographical plane uses the poles of Earth as a reference for the system, celestial coordinates systems use celestial objects as a reference. This results in the celestial coordinate systems often being rotated from the geographical system and they often differ in between as well. Therefore, when using celestial coordinates, it is important to specify which coordinate system is used. For that reason, when an object is located by a satellite or telescope, it is assigned some coordinates either in the form of (RA, DEC) or (l, b), or which coordinates system that has been used. Moreover, as celestial objects are in motion, a time also has to be given for the precise location of an object.

In this project, we will use two coordinate systems. We will use the Galactic Coordinate System and the International Celestial Reference System(ICRS). The Galactic Coordinate System is observed from the location of the sun[18]. The celestial plane of the coordinate system is parallel with the plane of our galaxy, making the galactic longitude, or galactic height, zero if looking at the plane of our galaxy. The galactic height ranges from -90 to 90 degrees, making the galactic poles perpendicular to the plane of our galaxy. Moreover, looking towards the center of our galaxy from the perspective of the sun, it has a right ascension of zero. The right ascension has a range from zero to 360 degrees[18]. Using the positioning of the sun removes the need for accurate time declaration as we move relatively slowly around in our galaxy. However, it is updated over time due to the tilt of the earth shifting[18]. This means a set time is decided, and today the time used is J2000. J2000 refers to epoch 2000, which is 1. January year 2000. The ICRS system point of reference is located at our sun like the Galactic Coordinate system. However, the ICRS system has its reference in several extragalactic radio sources[19].

The coordinates provided by SWIFT are given in RA and DEC meaning it is in the ICRS. Furthermore, coordinates from SWIFT follow J2000. When receiving the coordinates from SWIFT for the approximate location of the transient, NOT has to be moved towards those coordinates. NOT calculate with the use of its current position and time, together with the coordinates from SWIFT, the new position it should turn towards. When the position is found, NOT uses its auto-guiding system to keep the location of the sky locked[20].

When imaging is done by NOT, the data has been WCS calibrated. However, to get more accurate coordinates in RA and DEC of sources in the images from NOT, they have to be WCS calibrated again. To do this, PyNOT compares the initial WCS solution from the header with objects in the Gaia catalog. Then PyNOT fits the sources to give a more accurate WCS calibration.

2.2 Methodology for fast transient detection

In the following section, the thoughts behind the methods used in this report will be explained as well as the concepts behind them will be elucidated. This includes testing of PyNOT. Moreover, it also includes the extension of PyNOT for using other catalogs than Gaia and optimization of so. This section will also review how using information on source size(quantified by full-width half maximum(FWHM)) can be used for identifying and selecting point sources. Further, this section will review the possibility of using the fading of the transient between images for better transient detection. Moreover, easier usability by using a graphical user interface will be elaborated.

2.2.1 Using PyNOT for data reduction and transient detection

As mentioned in the introduction, PyNOT works very well and is easy to use directly from the terminal. However, for further development and testing, I found it easier to work with PyNOT in a combination of the source code and writing scripts that make use of PyNOTs source code. For that reason, this is how I have worked with PyNOT in this project. The PyNOT pipeline, which starts with raw images and continues forward to transient detection, consists of three steps. These steps are intended to be executed in a terminal. Yet, with some minor changes the pipeline can be run in a script by importing the *"initialize"*, *"run_pipeline"* and the *"find_new_sources"* functions from the *"main"*, *"phot_redux"* and *"transients"* folders respectively from the source code. After running the two first functions a reduced image is created like the one seen in figure 4 where GRB220101A is used as an example. After those two steps in the PyNOT pipeline, PyNOT has combined the OBJECT-images as well as BIAS and FLAT-field corrected. Moreover, it has WCS calibrated the image with the use of the Gaia catalog as a reference, as well as the image is corrected for cosmic rays. Further, source detection in the reduced image has been done with the SEP-library tool which PyNOT is using. If a source detected in the reduced image is too oval, PyNOT is sorting it away. However, in the reduced image in figure 4, it is very difficult to spot the transient which should be in the field and hopefully also one of the sources marked with a red circle as those are the sources detected.

In the *"phot_redux"* function, PyNOT is downloading a part of the Gaia catalog with known sources covering the area of the image plus a little bit of the surrounding area. Running the *"find_new"* function, one or two error circles appear depending on whether you give both BAT and XRT coordinates or only one of them. PyNOT then compares the sources found in the reduced image shown in figure 4, with sources in the Gaia-catalog. PyNOT only uses the coordinates of the sources in the Gaia catalog to compare whether or not there is a match. PyNOT does so by calculating the angular distance between the objects detected and the objects in the catalog, and checking if the calculated distance is under a threshold which by default in PyNOT is set to 1.5 arcseconds. PyNOT marks the sources to be in Gaia with a blue square. Moreover, PyNOT uses the error circles from BAT and XRT to tell if a source is consistent with either the BAT coordinates or both the BAT and the XRT coordinates. If a source is found to only be consistent with BAT, it is marked with a yellow circle. If a source is found to be consistent with both BAT and XRT it is marked with a green circle. The remaining sources are marked with red circles (Figure 5).

By default, PyNOT uses a magnitude threshold of 20.1 when looking for new sources and comparing them with the Gaia database. Anything fainter, meaning a number above the threshold, will be sorted out. In the example of figure 5, the magnitude is kept at default. As seen in the example figure 5, eight sources are found with apparent magnitudes brighter than 20.1 in the image of GRB220101A in the r-filter, which are not in the Gaia-catalog. In this particular example, only one source appears to be consistent with both the BAT and the XRT error circle. Those error circles are based on statistical errors in the coordinates provided by the BAT and the XRT. The errors are in PyNOT used as a radius and with a confidential interval of 90%, meaning the transient might be outside the error circle but still be consistent with BAT or XRT. This is also the case for the source labeled number 6 in image 5, where the source is found just outside the error circle of XRT, but still counted as consistent with it.

It might occur there are found several sources within the error circles of BAT and XRT as well as there might not be any transient candidate found. It might also happen there is no XRT coordinates available and we have to rely on the larger error circle of the BAT. If so, there tend to be more unknown sources in the field of the error circle as it is much larger. Therefore, it would be even more difficult to guess which of the candidates that are the transient. For that reason, it is interesting to test the strength and robustness of the functions PyNOT is using. This will reveal how well PyNOT does for transient detection and potential pitfalls. The test of this is carried out on five different GRB events and is presented in the results section.

2.2.2 *Expansion of PyNOT use of catalogs*

As PyNOT has quite limited methods to do transient identification, trying to expand upon PyNOT seems reasonable. For that reason, I have explored this option. I have looked into expanding PyNOT to use other catalogs as well as using several catalogs. There are two reasons this could be useful.

One of the reasons might be that we are detecting multiple sources within the BAT or XRT because the sources are not present in the Gaia catalog which PyNOT relies on. It is a fair assumption that the Gaia catalog might not be perfectly complete. Thus, it makes sense to also compare with several other catalogs for a more complete overview of already known sources. Besides, soon the NOT will be upgraded with a new instrument for transient detection, named NOT Transient Explorer(NTE). This instrument can take images in the infrared. It is practical to take images in the infrared as infrared light in general is not exposed as much to extinction as a result of dust as optical light is [21]. Thus, being able to use catalogs covering sources in the infrared would be valuable in the near future. For this project, I chose to expand PyNOT to make use of the WISE and 2MASS catalogs as those are looking further into the infrared wavelengths than Gaia[22][23]. Based on Gaias $''edr3''$ catalog, Gaia is covering wavelengths around 330-1050 nm [22] which corresponds to the optical light between $0.3\mu m$ and $1\mu m$ [24]. Meanwhile, the WISE has four bandpasses at 3.4, 4.6, 12, and

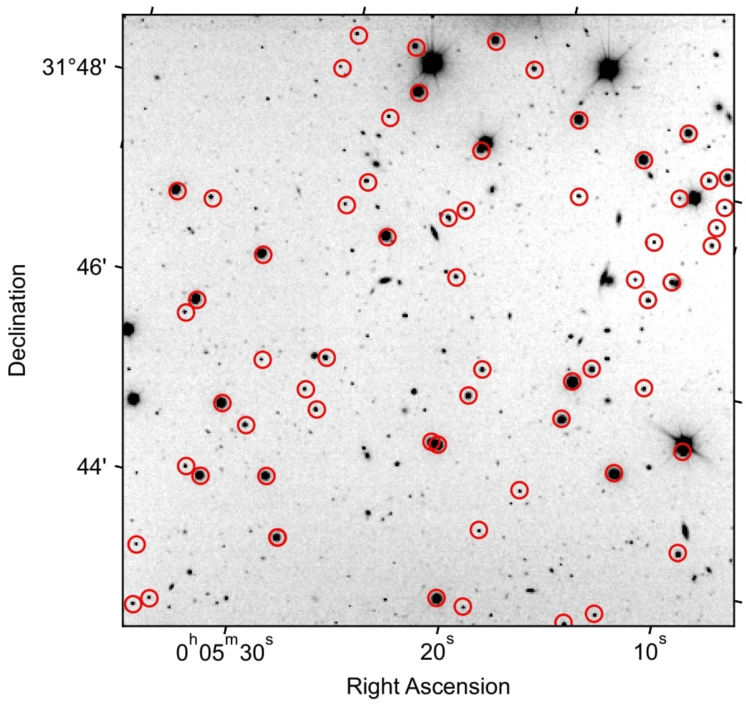


Figure 4: *This figure shows the reduced image from PyNOT of GRB220101A. The red circles are marking the sources that are found in the reduced image by PyNOT with minor/major-axis relation higher than 0.8.*

22 μm, together covering most wavelengths within that range[23], and hence covering most of the mid-infrared between 2.4μm and 20μm [25]. The 2MASS has three bandpasses at 1.235, 1.662 and 2.159 μm [23], covering most wavelengths within this range, making it look in the near-infrared between 1μm and 2.4μm [25]. The bandpass coverages of WISE and 2MASS are shown in figure 6.

It should be noted that objects can be observed across many wavelengths. This means that even though an image is taken in a particular filter with a specific range of wavelengths, catalogs that do not cover the exact same wavelengths might still provide a slightly better overview if it is used in addition to a catalog that does cover the wavelengths of the image. Some could argue that using more catalogs is better. However, this is not necessarily true as each catalog has to be queried at a remote server. Because of this, it takes additional time for each catalog and time is crucial for very fast transient detection. Many of the catalogs are having overlaps, thus, they are not necessarily contributing to a more complete overview.

Now, in the course of making this work, I have worked directly on the source code of PyNOT. It had to be changed such that PyNOT was also able to query the WISE and the 2MASS catalog if requested. This way, all three catalogs can be chosen for transient detection. I worked directly in the source code for this part due to PyNOTs potential for being used with the NTE. If NTE is going to make use of PyNOT it is not very practical that NOT can use the full pipeline, while NTE only can use some, and has to rely on another library as well. It makes more sense and it is more practical to make PyNOT more flexible, such that it can work fluently with both NOT, NTE, and maybe other

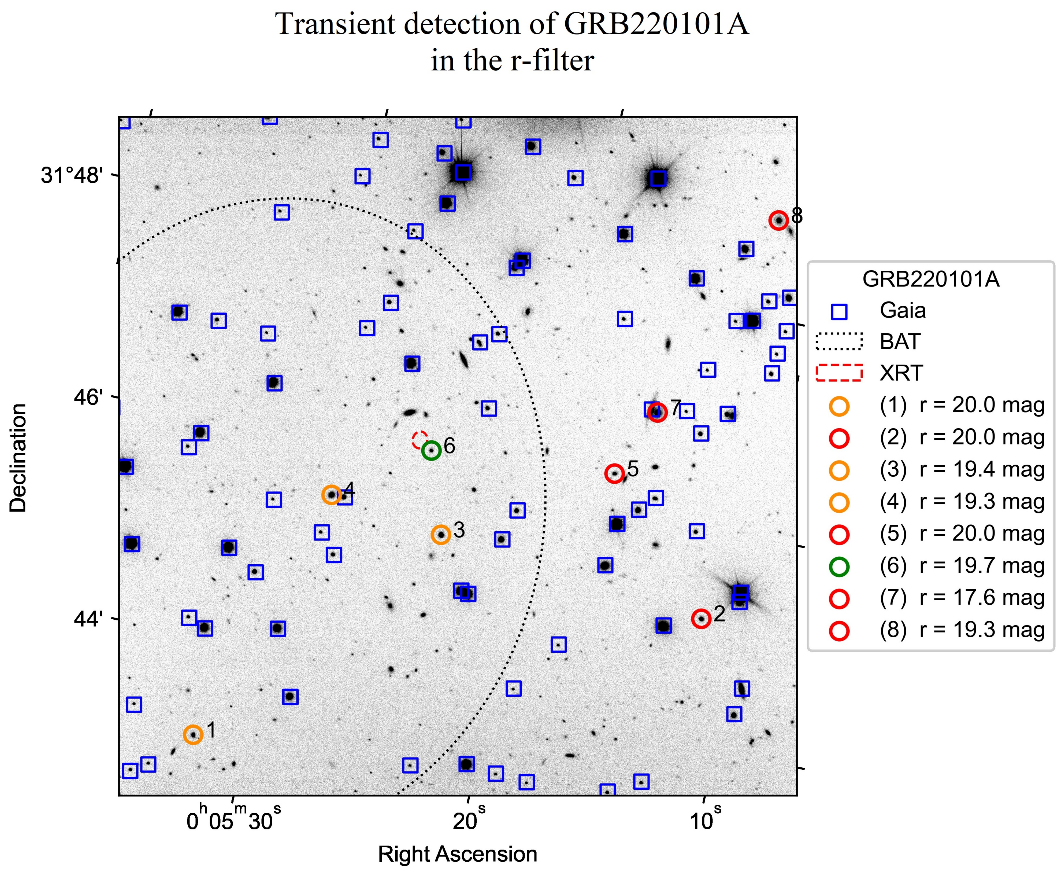


Figure 5: *This image is from transient detection with the use of PyNOT on the data set for GRB220101A. The image shows a potential transient detected marked with a green circle as it is both consistent with BAT and XRT. The transient detected is labeled number 6, with an apparent magnitude of 19.7. In the transient detection for this image, the threshold for magnitude in PyNOT is not changed, and hence the default 20.1 is used. The figure also shows which sources that are already in the Gaia catalog and they are marked with a blue square. Moreover, three sources are marked yellow as they are consistent with BAT but not XRT. Four sources are marked red as they are not consistent with BAT or XRT.*

future telescopes.

The transient detection function has been changed such that it compares with all three catalogs and creates three different images, one for each catalog. Cross-matching between catalogs is applied using the coordinates of transient candidates in each image and supplied to the observer. Thereby, he or she can see if and in which catalog has a matching source. The PyNOTs transient identifier function creates a table with information on the sources which are not found in the catalog used, which are candidates for being transients. An example is shown in figure 7 of GRB220101A, where PyNOT has compared the sources found in the reduced image against the already known sources in the Gaia catalog. In the table in figure 7, the WCS coordinates are those PyNOT have derived the sources to be at, not the coordinates in the catalog. A table like the one in figure 7 is generated for all three catalogs. Thus, the three catalogs can be cross-matching through these tables. This potentially reduces the number of transient candidates. However, a source should not be discarded because it is found in a catalog, but the prioritization should be lower, as it might turn out it is laying behind or in front of another object in the sky that is not in the other catalogs.

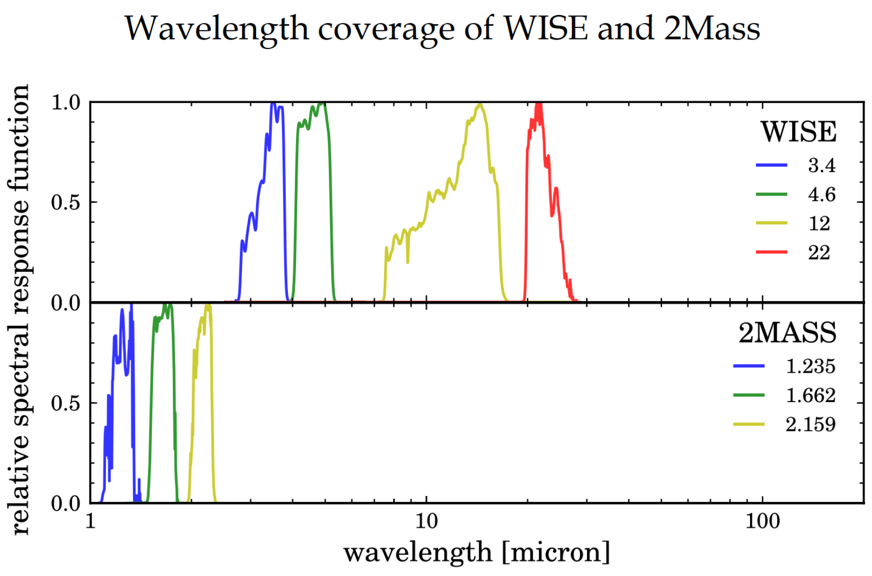
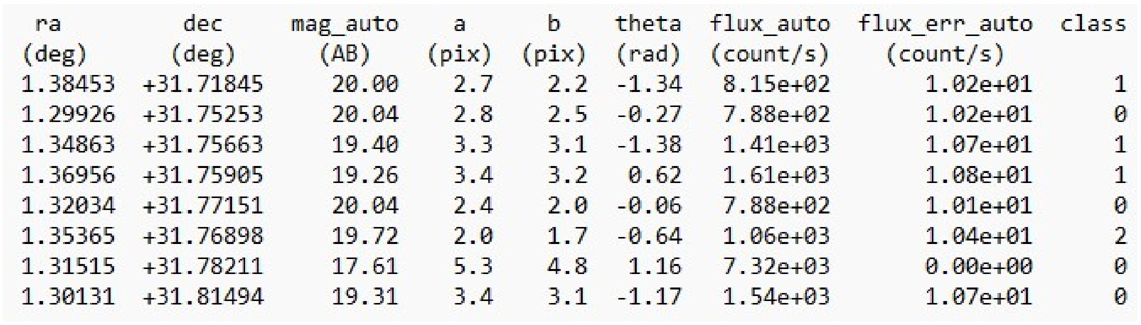


Figure 6: *In this figure the covered wavelengths of WISE and 2MASS are illustrated. As seen, WISE has four bandpasses that cover most of the mid-infrared wavelengths between* $2.4\mu m$ *and* $20\mu m$*. It is also shown that 2MASS is covering most wavelengths in the near-infrared between* $1\mu m$ *and* $2.4\mu m$ *with its three bandpasses. Together, the catalogs have a total coverage span from* $1\mu m$ *up to* $20\mu m$ *[23].*



```
  ra        dec      mag_auto   a     b    theta  flux_auto   flux_err_auto  class
(deg)      (deg)       (AB)   (pix) (pix)  (rad)  (count/s)     (count/s)
1.38453  +31.71845    20.00    2.7   2.2  -1.34   8.15e+02       1.02e+01      1
1.29926  +31.75253    20.04    2.8   2.5  -0.27   7.88e+02       1.02e+01      0
1.34863  +31.75663    19.40    3.3   3.1  -1.38   1.41e+03       1.07e+01      1
1.36956  +31.75905    19.26    3.4   3.2   0.62   1.61e+03       1.08e+01      1
1.32034  +31.77151    20.04    2.4   2.0  -0.06   7.88e+02       1.01e+01      0
1.35365  +31.76898    19.72    2.0   1.7  -0.64   1.06e+03       1.04e+01      2
1.31515  +31.78211    17.61    5.3   4.8   1.16   7.32e+03       0.00e+00      0
1.30131  +31.81494    19.31    3.4   3.1  -1.17   1.54e+03       1.07e+01      0
```

Figure 7: *This is the information table generated by PyNOTs transient detection function. It provides information about the transient candidates in order of their labeled numbers. In other words, the first row corresponds to the source labeled 1 in the corresponding image, and so on. The table provides information about the position, magnitude, flux, and corresponding flux errors for each source. Moreover, the table has four titles in the table header row that are not self-explanatory. Those are "a", "b", "theta" and "class". The "a" and "b" is the major and minor axis of the source measured in pixels and theta is the angle at which the ellipse is rotated. The "class" is a number indicating if the source is consistent with BAT and XRT. If a source is consistent with only BAT it gets the value "1" in the table. If it is consistent with both BAT and XRT, the source is given the value "2". If a source is not consistent with either BAT nor XRT it gets the value "0". As it can be seen, the sixth source is labeled "2" indicating this is most likely the transient.*

2.2.3 Potential problems and test of those

As previously mentioned, it may occur that a transient has coordinates that are within a too short angular distance to an already known source in one of the catalogs and it may not be found. This happens for GRB220101A, where the transient is very close to a source in the WISE catalog(Appendix 3). This has also been noted in the paper by GCN Circular 31373 [26], that there is a source of only 1.8 arcseconds from the transient.

To test whether this is a huge problem I have generated coordinates within a query region, to simulate transients. For those coordinates, I find the angular distance to the coordinates that are in the catalogs of the same region. If the distance is lower than the threshold in PyNOTs source code it is considered a match. To calculate the angular distance I use *"pyasl.getAngDist"* from PyAstronomy which is a library in Python. *"pyasl.getAngDist"* is making use of a Haversine function to calculate the angular distance, which is shown in equation 1. The Haversine function in equation 1 is written as it is in the source code of the *"pyasl.getAngDist"* function[27].

$$d = \frac{2 \cdot arcsin\left(\sqrt{sin\left(\frac{(dec1-dec2)\pi}{180\cdot 2}\right)^2 + cos\left(\frac{dec1\cdot\pi}{180}\right)\cdot cos\left(\frac{dec2\cdot\pi}{180}\right)\cdot sin\left(\frac{(ra1-ra2)\pi}{180\cdot 2}\right)^2}\right)}{180\pi} \tag{1}$$

This way I simulate how often a transient will appear close to another already known source, and how strong the method of only using coordinates is. This though, may depend on where in the sky the image is taken, as some regions of the sky are more dense in amount of stellar objects. Especially, when looking in towards the center of our galaxy compared to looking at the galactic poles. This, I test by doing as just mentioned. However, I carry out the test at different heights above the galaxy plane as well as I do it at different right ascensions. This is expected to be comparable to the distribution of the number of stars found at a certain declination as it should be proportional to the density of objects.

2.2.4 Source size (FWHM)

When taking images like the one shown in figure 4, we observe a variety of different stellar objects. Most of them will be stars but some of them might be galaxies and some might be a third event like a transient. The light distribution of galaxies will be flatter and wider compared to stars or transients which will be narrow spikes. This spike is due to the extremely small width to distance ratio of the object making it so small in an image that it can not be resolved. Whether an object can be resolved in the image depends on the proper size of the object, the distance to it, and the seeing. For ground-based telescopes, this is mostly affected by the seeing. As long as the observed width of an object is larger than the seeing, we can resolve it. For the location of NOT, the most common seeing is found to be around 0.6 arcseconds per pixel [28]. Making figure 8 based on the cosmological model, shows the proper size of the object if it fills 0.6 arcseconds in the image. The figure helps us to see how far out we would be able to resolve a typical spiral galaxy. A spiral galaxy has a diameter of 5-100 kpc [29]. It can from the plot be seen if a galaxy is five kiloparsec or more it would in an image out to a distance of z=2 fill more than the 0.6 arcseconds, making it larger than the lower limit of what can be resolved. Most of the galaxies would be larger than this as it is the smallest spiral galaxies that are five kpc in width. However, it should be noted very compact galaxies do exist, and those we may not be able to resolve that far out [30]. Further, at high redshifts galaxies appear larger as seen in the model in figure 8. This effect is due to the universe expanding. However, galaxies also tend to be much smaller at those high redshifts. In a report by L. Yang et al.[31], they determined the size of 19 bright galaxies at redshifts between five to nine to be much smaller than a typical galaxy. Most of them were determined to be about one kilo parsec in radius [31]. This suggests that the model in figure 8 can not be used at high redshifts and that galaxies at high redshifts can not necessarily be resolved either. The equations and the script for the model are presented in Appendix 5.

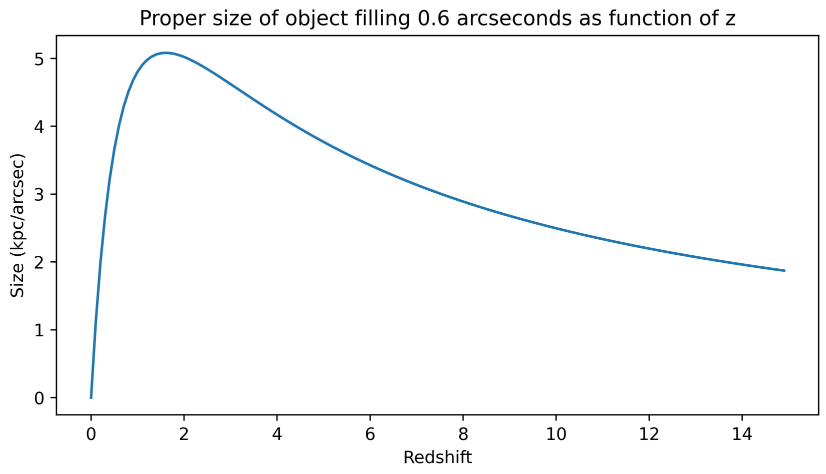


Figure 8: *In this figure a CDM-model is presented showing the proper size of an object filling 0.6 arcseconds in the sky as a function of redshift. The model uses* $c = 2998000\frac{km}{s}$, $H = 70\frac{km}{Mpc \cdot s}$, $\Omega_m = 0.3$, $\Omega_\Lambda = 0.7$ *and a curvature constant of* $\kappa = 0$ *[32]. From the model, it can be depicted that a galaxy larger than five kilo parsecs can be resolved in an image taken with seeing at 0.6 or less, even at high redshift.*

For this reason, it is possible to differentiate between stars and most galaxies. Thus, stars and transients will be referred to as point sources, while everything else is expected to be a galaxy. However, a point source will in practice not be an extremely thin spike but instead, be a sharp peak with some width as there will be some smearing due to atmospheric distortions. The smearing of the peak is proportional to the seeing. Nonetheless, a point source will still have a sharper peak than a galaxy because both will be exposed to the same effects.

In an image, just like the one in figure 4 or figure 5, it is difficult to distinguish a point source from a galaxy. To make this differentiation between point sources and what is not, it is useful to look at the FWHM of each object in the image. FWHM is a measure of the width of the light distribution at the point where it is half of the maximum height of the peak. This is illustrated in figure 11, where a 1-dimensional Gaussian fit is made on the source labeled number 3 from figure 5. Point sources all have the same initial width in the light distribution, as they are too small to be resolved in an image. For that reason, point sources will also have almost the same FWHM. Whereas, galaxies that can be resolved will have varying FWHMs as they also vary in size. The deviation of FWHM for a 1-dimensional Gaussian comes as the following:

$$f(x) = A \cdot exp\left(-\frac{(x-\gamma)^2}{2\sigma^2}\right) \tag{2}$$

Here, A is the normalization constant or the amplitude $\frac{1}{\sqrt{2\pi}\sigma}$. As we want half of the maximum height, which is the same as the amplitude, we divide A by two and set it equal to the Gauss function, continued by solving for x.

$$\frac{A}{2} = A \cdot exp\left(-\frac{(x-\gamma)^2}{2\sigma^2}\right) \tag{3}$$

$$ln(1/2) = -\frac{(x-\gamma)^2}{2\sigma^2} \tag{4}$$

$$ln(2) = \frac{(x-\gamma)^2}{2\sigma^2} \tag{5}$$

$$x = \pm\sqrt{2ln(2)}\sigma + \gamma \tag{6}$$

FWHM = $(x_+ - x_-)$ as we want the full width from the middle at half maximum. Thus,

$$FWHM = (x_+ - x_-) = \sqrt{2ln(2)}\sigma + \gamma + \sqrt{2ln(2)}\sigma - \gamma \tag{7}$$

$$FWHM = 2\sqrt{2ln(2)}\sigma \tag{8}$$

As seen in equation 8, from the deviation above, the FWHM of a Gauss function can be found solely from the σ parameter. This method of fitting and finding the sigma fit parameter and using equation 8, can then be applied to any source of interest and reveal its FWHM. If a reference for FWHM is established for point sources, the FWHM found for an object can then be used to distinguish if the object is a point source or a galaxy. However, the atmospheric distortion is not the same every day. Therefore, neither is the smearing and a value for FWHM, telling when a source of interest is a point source or not, can not simply be determined nor used across different images. However, as mentioned, we are expecting a variety of stellar objects in each individual image. We also expect the seeing to be the same for all the sources in the individual image taken. As a consequence, a fit and calculation of FWHM can be done for all of the sources in a given image to make a sample containing two distributions, which each likely will follow Gaussian distributions, together following a double Gaussian. Testing this upon the image of GRB220101A supports it is a good assumption as seen in the upper plot of figure 9.

Fitting a double Gaussian to the sample distribution and separating the parameters into two separate Gaussian functions and fitting those to the sample, it is expected there would be two distributions. It is expected to see a sharp and narrow distribution containing the point sources and a much more spread out distribution containing galaxies with a broad span in FWHM. This is also illustrated in figure 9, where the image of GRB220101A is used. The very narrow peak would be the point sources, meaning anything that has a FWHM that does not belong to this distribution is most likely not a transient which is very valuable information. However, as the galaxies can have a wide span of FWHMs, we can not exclude that a galaxy is in the distribution of the point sources. This is seen in the bottom figure in figure 9 where there is an overlap between the distributions. I have chosen not to include the sources of interest in the sample created for FWHM comparison, as the sample then can be used as a baseline not biased by the sources to investigate.

Even though a source appears to not be a transient because its FWHM does not lay inside the narrow distribution, it should not be totally ignored. It could be that it is a transient inside of a galaxy which we are able to see in the image as well. We can instead use the information to lower the priority for

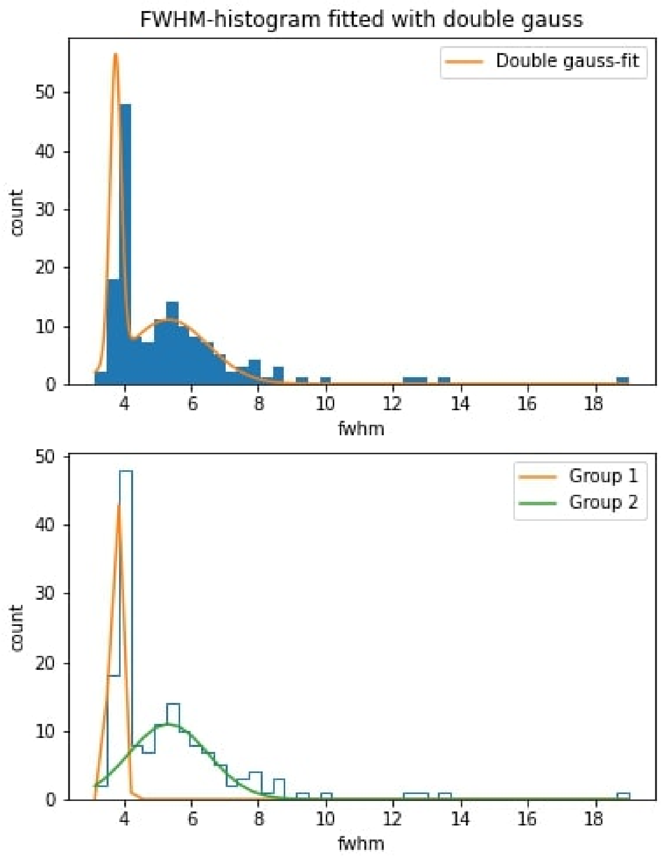


Figure 9: *The figure in the top shows a good fit with a double Gaussian to the FWHM sample created over the sources detected in the image of GRB220101A. In the bottom figure, the double Gaussian has been split into two distributions of sources. This makes it clear that two distributions are present. For this histogram, 45 bins are used.*

this source. Thus, list the information whether it is a point source or not in a table together with other relevant information for the person investigating to review.

A problem that may arise here, is that automated fitting can be difficult to make robust. This is especially true if the automated fitting is done to a histogram made with a relatively low number of data. Hence, as the number of sources in an image might be relatively low depending on the position of the sky in which the image is taken, automated fitting can be troublesome. It is often the case that the number of objects is lower in an image if it is taken towards the galactic pole or at high right ascension. A small change in the number of bins for the histogram can make a huge difference as shown in figure 10. In this figure, the number of bins is 47, which is only 2 more than the 45 used in figure 9, where the fit worked very well. Two solutions for this problem are available. One is to make the plots of the fitting come together with the images of transient detection and the information table, such that the person inspecting the images easily can see if the fit is good. The person observing can then judge if the information in the column for whether a source is a point source or not should be trusted. With this solution, it may happen that there are cases where the information has to be disregarded due to the fit being useless as shown in figure 10. Another solution that is much more robust is using the information that the point sources in a histogram will be stacked and therefore most likely be those defining the tallest peak, which is also very clear being the case in the histogram of figure 9. Then the FWHM of the tallest peak, plus-minus 25%, can be used to tell if a source is a point source. The reason for the tolerance is that in practice, the point sources will not be exactly the same, even though they are very close. Moreover, the histogram is also depending on the number of bins and the bins will have a width. Setting a relatively high tolerance is also better than setting it too low, as it is important to not sort out point sources. This method rests on the assumption that the image is not highly dominated by non-point sources, which is a fair assumption for a ground-based telescope taking a broad picture of the sky. The last method is the one I used as I in this project prioritized the method which is most robust although none of them is 100 percent secure. The first method overlap in its distributions and are dependent on a good fit which may fail, and the other method has an arbitrary choice of tolerance being its biggest weakness.

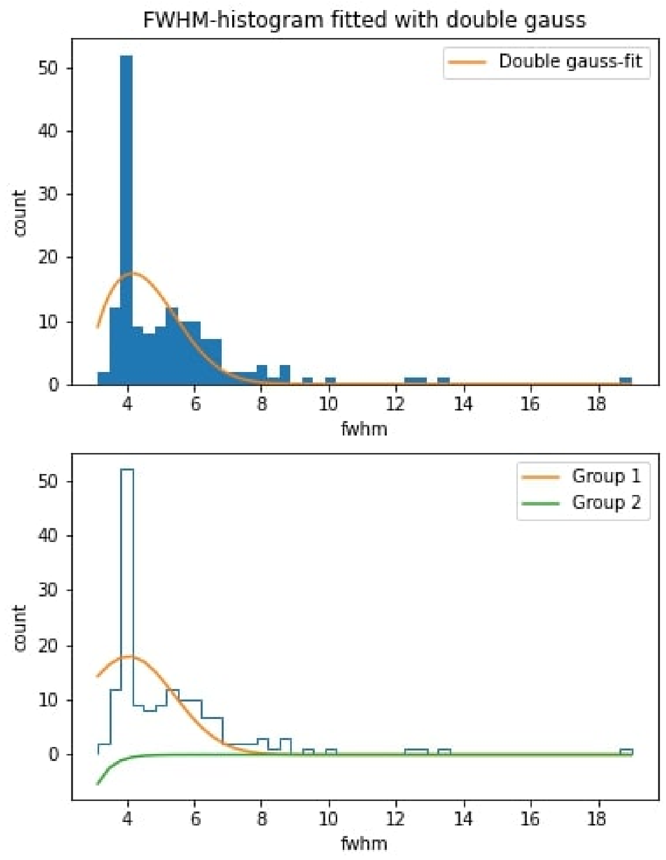


Figure 10: *The figure in the top shows a fit with a double Gaussian to the FWHM sample created over the sources detected in the image of GRB220101A. In this figure, the number of bins is 47 instead of the 45 bins used in figure 9. This small difference results in a fit that does not show two distributions when being split into two Gaussian functions as shown in the bottom figure. Holding this figure together with figure 9 shows the challenges with automatic fitting as it is sensitive to the number of bins used.*

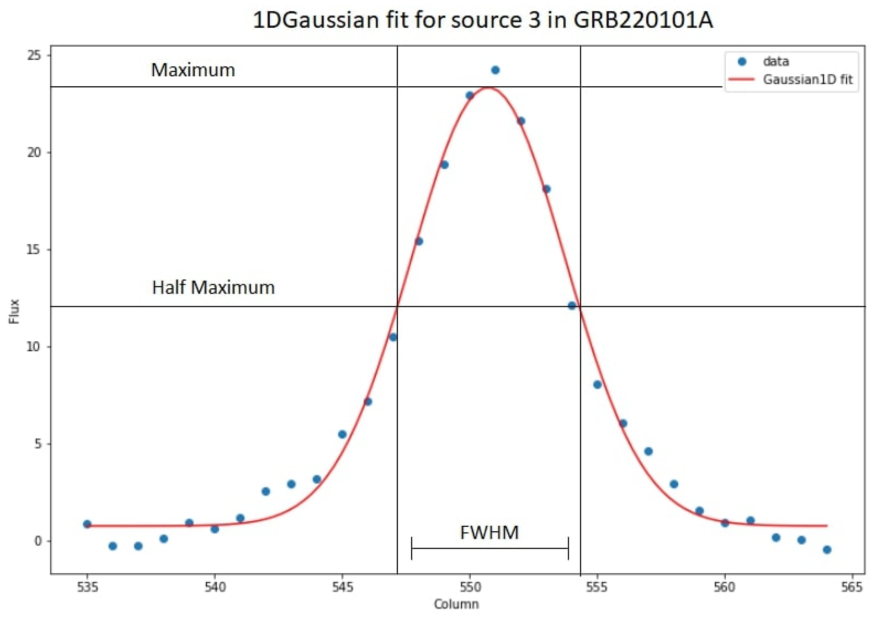


Figure 11: *In this figure, estimating FWHM is demonstrated on the source labeled "3" in the image of GRB220101A. As shown, the FWHM is the full width where the peak is at its half maximum. In practice, a function is fitted to the data, and the FWHM is taken of this function. In this example, a Gaussian function has been fitted to the source labeled "3".*

2.2.5 *Detecting fading sources*

When it comes to GRBs, they are well known to fade rather quickly. As mentioned in the introduction, the light curves are decreasing after an early break following a power law [13]. Looking for this fading can be used to identify the transient if several images are taken. It is often the case that several images are taken sequentially in time. Looking at differences in magnitude between the images can tell if an object is fading away. The magnitude of a transient is expected to decrease following equation 9. Here $t1$ and $t2$ are the times after the event. When imaging, it would be the average moment of time when image one and image two were taken after the event erupted. It is the average time as the images are taken over a set exposure time. Hence, this would be a better estimate than using just the beginning of the exposure time. Alpha is a parameter that varies from event to event depending on how fast the transient is fading.

$$drop = 2.5 \cdot \alpha \cdot log10 \left(\frac{t1}{t2} \right) \tag{9}$$

When looking at the differences in magnitudes between images, it is interesting to investigate if any of the objects follow this theoretical drop. If so, it might be the transient. Any significant drop in magnitude is also a huge clue even if it not follows the expected drop perfectly. It might also be alpha has not yet been estimated, as this is often done by fitting to the light curve, which requires we know the source to begin with. Because of that, a significant drop is the only clue for fast transient detection. But the results in this report will still be tested against the theoretical drop as it will indicate to which extent measuring the drop is possible. Nevertheless, to measure a significant drop, the errors when doing error propagation have to be less than the drop before we can be sure it actually is a drop. Otherwise, the apparent drop could just be noise. Hence, we desire small errors or a large drop. From equation 9, it is evident that the smaller the fraction is, the larger is the drop in magnitude as well. A smaller fraction can be achieved by taking the images with more time in between. It is unfortunately not convenient to waste time just waiting while doing very fast transient detection. For that reason, it is important to instead take the images as early as possibly such that the relative difference between $t1$ and $t2$ is large, leading to a smaller fraction as well.

For that reason, I use data from GRB220521A. The images from GRB220521A should be adequate as they are taken quite early after the event was first detected. For the data set I got, there are three out of four of the images in the r-filter in which the transient is observable. The images are taken 17.2, 22.6, and 27.4 minutes after the event first was detected. With the data for this event, I test if it is possible to observe a significant drop in the images from NOT, and if so, also see if it follows the theoretical drop.

In practice, I try out three different methods. For all of them, I use PyNOT for the data reduction of the images. Normally when a data set is given, PyNOT uses the average of the images to generate one final image, which for this set would have been four images. Hence, I need to split the data set into separate data sets. For this reason, I created a script that copies all the files with the exception of the OBJECT files except for one of them and sort it into a new folder. This is looped to cover each individual OBJECT image. This way I end with four data sets with just one OBJECT image in each. Those data sets are then fed to PyNOTs data reduction pipeline. After cross-matching the sources between images, calculating their magnitudes with equation 10 and there respectively errors with equation 11, analysis of the data can be done to see if there should be any drop. More specifically, I establish an average magnitude over the sources brighter, or around the same magnitude, than the transient. I do so, as it is fair to expect the other sources to be stable, over such a short period of time, and a baseline is needed for each image as they may vary slightly. This baseline gives something that relates the images with each other. The transient can then be subtracted from the respective means. This should provide me with three values, as there in the case of GRB220521A are three images I use as the last image does not have the transient visible. The difference between these values should then be the measured drop, such that value two minus value one equals the drop in magnitude for the transient between their respective images. The same goes for the second and the third image and between the first and the third image. These drops will be compared against the theoretical drop. Together with their errors, it will be judged if the drop is measurable. The fourth image can not be used in the test due to it not having a magnitude measured. So even though it of course is a big clue an object disappears, it is not relevant for this specific test for detecting fading objects.

$$m = ZP - 2.5 \cdot log10(flux) \tag{10}$$

$$m_{err} = \frac{2.5}{ln(10)} \cdot \frac{flux_{err}}{flux} \tag{11}$$

As mentioned, I will try detecting the fading in different ways. First, I will try using PyNOT to get the values for the fluxes and the flux errors and then cross-matching the sources so I only use sources that are present in all of the images. However, this will turn out to sort away many sources due to the area of GRB220521A being very crowded and cross-matching then quite hard because of the limiting precision of WCS calibration. The second method I will try is cutting each image around the same

WCS-calibrated XRT coordinate, such that each image should have pretty much the same center. I will then use SEPs object detection to detect objects and create apertures in the first image. These apertures will be used in the other images, making the cross-matching redundant which should let me keep more sources. This leaves a systematic error related to the WCS calibration, but it is taken into account as I establish a baseline in each image. In the last method, I will be using a point spread function(PSF) for flux and error detection. This has the potential to reduce the errors as PSF uses fitting instead of large apertures.

2.2.6 Automatisation and usability

PyNOT is already semi-automated in the sense, you only have to write three commands with some parameters to make it work. Despite that, I believe making a graphical user interface(GUI), which makes use of PyNOT combined with some extra scripts as back-end can heighten the usability. This applies especially to a user sitting in the middle of the night, who is not familiar with PyNOT or maybe has little experience in Python. If using other functions like point source selection, detection of fading objects, or maybe just further investigation of an image, then a GUI will be a tool to gather those in one package if developed.

When creating a GUI it is important that it is very easy to use and it is manageable. The scope of the GUI is also important to establish. The GUI I will create for automated transient detection should take a few inputs in the front-end and do all the pipelines in the back-end. This includes the pipelines from PyNOT as well as those pipelines I create in this project. It should at the same time not have any noticeable impact on the performance of the data processing in the back-end as the time is crucial for fast detection. Thus, if this requirement is not met, the purpose of the GUI is not worth it as it defies its initial purpose. The GUI should also present the data in a manageable way. For this to be accomplished, I decided the GUI should be built up of several different layers. Figures of those layers are presented in the result section in figure 30, 31, 32, 34, and 33.

For the development of the GUI, I have decided to use Python, as the back-end code is written in Python. For the front-end part, I will make use of Pythons library PyQt5 together with QtDesigner. QtDesigner is a program that helps with the designing of the layouts as it has several functions such as drag and drop of elements. When a layer is created in QtDesigner, it is saved as a UI-file. This UI-file can be imported in Python or directly translated into Python code from a Power shell with the command: pyuic5 –x ”filename”.ui –o ”filename”.py [33]. When the layer is used in a script, input cells, labels, and so on can be used or changed like any other Python object.

I find that four layers are sufficient for this GUI. First, a window that shows up when you initialize the GUI. This layer should only consist of the most necessary options, which the user has to manually act on. The window should have the option to choose whether you have already run the PyNOT-pipeline on your own. In this way, you will have the option to only be presented with the data and not have to run the whole pipeline again, saving some time. Therefore, this checkbox is more for when someone wants to review data already processed before. You should also be able to find the path of your raw data from here, such that the program is not dependent on being in a specific location. The rest of the inputs should be input parameters, like the one you would use if you ran PyNOT in a terminal.

The front page should preferably also have a pop-up window telling you if some necessary input is missing.

Furthermore, there should also be an information window, which presents the information from the pipelines in a convenient way. In the case of data from PyNOT, it makes sense to have a place where the plots are presented alongside the associated tables with relevant information. As a lot of data sometimes is generated, it is not possible to have it all displayed at the same time. Hence, it is necessary to have the option to switch between images and associate tables, based on catalog and filters and an option to only see those marked as point sources.

In between the front page and the information window, it would make sense to have a small window showing how far in the process the pipeline is, preferably with a timer for the user to follow.

The fourth window would be a small pop-up window containing the most basic and important information found when compared with the Gaia catalog in the r-filter. This window is smaller as it should be looked at as something for a very quick overview. This window could be color coded in the table, such that a red color is against it being the transient while a green color indicates it might be the transient. This way it should be fast for the eye and easier for the observer to quickly make a judgment upon which objects should have the most attention, without disregarding any.

2.2.7 Optimization

Doing fast automatic detection of transients require scripts, and it requires they are fast. For that reason, it is often necessary to optimize the code by changing its structure. It is quite natural to use for-loops and do sequential execution, but not very often optimal for minimizing run time. Instead, it is better to make use of the Numpy library in Python as it is optimized to use arrays for parallel execution. Besides, the Numpy library has plenty of mathematical functions, such as array and matrix operations, which are implemented in pre-compiled C code which makes it much faster as well as easier to work with [34].

Moreover, it might happen the task to be executed is not CPU bound, but instead Input/Output(I/O) -dependent. In this case, the CPU is not working at its maximum capability, as the bottleneck is elsewhere. This appears, among other things, when requesting data from a remote server that takes time to return the information. In Python where the tasks are executed sequentially, the CPU will simply wait for the data being received before doing the next task. This is inefficient especially if several things are requested. Therefore, I introduce multi-threading in Python and the library $"threading"$. Every modern CPU has cores and threads. The cores can be seen as the number of workers the CPU has, and Python is by default only using 1 core. However, Each core has several threads, which is the capability to handle several tasks at the same time within the core. This means the CPU can run several tasks in parallel. This does not make the CPU faster. Hence, it is thus not useful in CPU-intensive tasks. Nonetheless, it is useful when tasks are I/O bound, as the CPU in the meantime can continue on other tasks, using its full potential.

In the expansion of making PyNOT use other catalogs for transient detection, we have an I/O-bound process three times as we want to request the Gaia, the WISE, and the 2MASS catalog. For this reason, the implementation of threading has been made in PyNOTs source file, "$WCS.py$", as this is where the Gaia-catalog initially is requested. The implementation of threading is illustrated in figure 12. For it to work, the source code of PyNOT has been changed such that we spawn two new threads, one to query the WISE catalog and one to query the 2MASS catalog. As the Gaia catalog usually is the fastest to query and also the one used to WCS calibrate the image, the initial main thread will query the Gaia catalog and afterward continue the data-processing inside the *correct_wcs* function in the ”*WCS.py*” file. This occurs while the other threads in the meantime are collecting the two other catalogs. The threading library has a *"join"* function, which waits for the threads to finish and joins them. This function is used at the bottom of the ”correct_wcs” function to make sure we have got all the catalogs before moving on. This is simply to avoid errors as the catalogs have to be downloaded before using some of the functions that are used later on when doing transient detection.

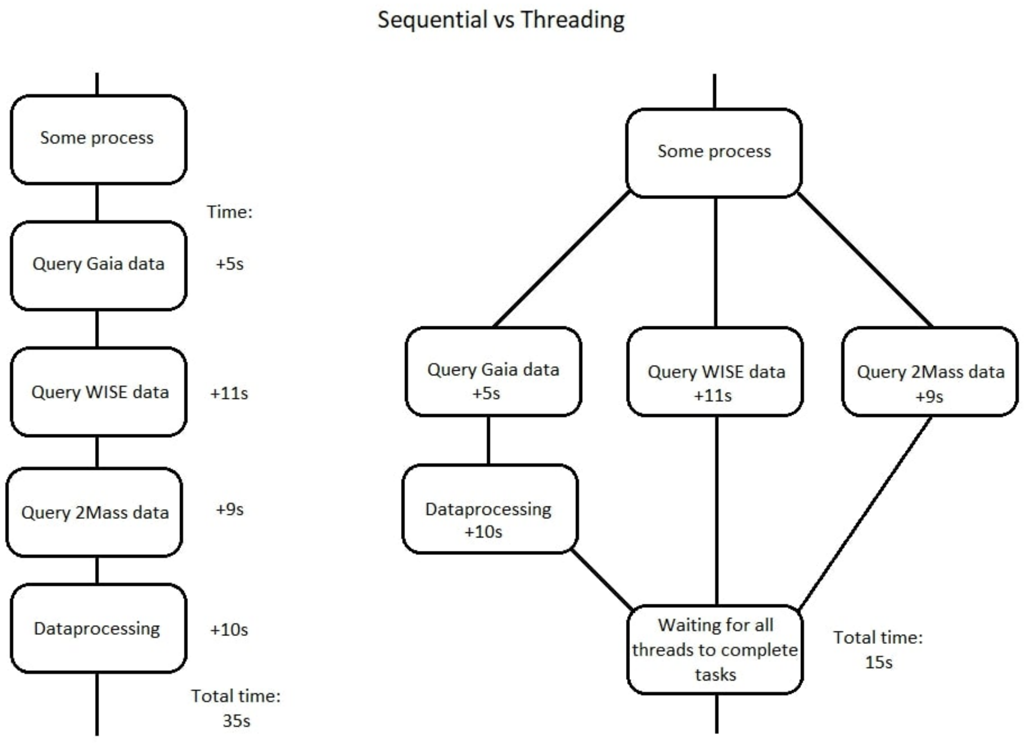


Figure 12: *This is an illustration of using sequential execution versus multi-threading. The times are illustrative but realistic times. This choice of using illustrative times instead of real runtimes is because the times vary from run to run, and illustrating the concept is more important. Therefore, the times have been chosen such that they reflect actual observed times, but they should not be taken as results. This illustration shows that running I/O-bound tasks in parallel with the use of multi-threading can provide a significant speedup over sequential execution.*

Creating new threads is done with the following. First, the threading library is imported. Then *"threading.thread"* is used to create a class object that takes the function to be run in the thread and its arguments as inputs. The thread object can be started with the start function as well as the join functions waits for the task to finish.

```
import threading

new_thread = threading.Thread(target=TargetFunction, args=(FunctionsArguments,))

new_thread.start()

new_thread.join()
```

3

DATA

3.1 Introducing data

My work is based on imaging data from the NOT. The files I have worked with are raw data in the format of FITS(Flexible Image Transport System)-files gathered in a folder with the name indicating which GRB event we are processing. The FITS files are provided from the Alhambra Faint Object Spectrograph (ALFOSC). The datasets I have worked with are from GRB190114A, GRB190114B, GRB190114C, GRB190829A, GRB191019A, GRB211211A, GRB220101A, and GRB220521A.

The raw FITS files provided consist of BIAS-frames and FLAT-field images together with OBJECT files. Each FITS file has a header indicating which type of file it is, the time it is taking, the direction of the telescope, exposure time, and other useful information.

-**OBJECT-files** are images taken of the sky in an area in which the transient of observation is expected to be within based on information from the BAT and XRT coordinates provided by SWIFT. The images may vary in exposure time. Those files I have worked with, vary between 120 and 300 seconds exposure time.

-**BIAS-frames** purpose is to subtract and thereby adjust for the readout-noise of the telescope. The frames are created by taking zero-second exposure images with the shutter of the telescope closed[35].

-**FLAT-fields** are images taken of an evenly illuminated surface. They are used to calibrate the response of the individual pixel in the CCD as this may vary across the CCD.

As stated on the webpage of NOT, I should include the following statement:
"The data presented here were obtained with ALFOSC, which is provided by the Instituto de Astrofisica de Andalucia (IAA) under a joint agreement with the University of Copenhagen and NOT."
(http://www.not.iac.es/instruments/alfosc/alfosc-acknowledgement.html)

3.2 Overview of the data used in this project

The data worked with in this project are from the events of GRB190114A, GRB190114B, GRB190114C, GRB190829A, GRB191019A, GRB211211A, GRB220101A and GRB220521A. In this section, an overview of the data gathered in each data set will be presented. This includes the number of BIAS, FLAT and OBJECT files. Moreover, it includes the time between the event triggered BAT and the first observation. Moreover, the times of exposure and filters used are presented as well.

3.2.1 GRB190114A

The data set from GRB190114A consist of three OBJECT files, 22 BIAS files, and 12 FLAT files. All three OBJECT images are taken with the r-filter and a exposure time of 300 seconds was used for all three images. The imaging started at 19:48:40 14th of January 2019, the same day as the event. The average times of exposure after trigger for the three images are 997.19, 1002.62 1008.06 minutes respectively. This corresponds to more than 16.5 hours after the first trigger by BAT.

3.2.2 GRB190114B

The data set from GRB190114B consist of 14 OBJECT files, 22 BIAS files, and 12 FLAT files. The OBJECT files are taken in the g, r, i and z filter. Three of the files are in the g-filter, three in the r-filter, three in the i-filter, and five in the z-filter. The imaging started at 01:46:15 15th of January 2019 the day after the event triggered BAT. This means that the first image has an average time of exposure of 6.27 hours after the trigger. The exposure times for all the images, with the z-filter images being exceptions, is 300 seconds. The images taken in the z-filter have exposure times of 200 seconds.

3.2.3 GRB190114C

The data set from GRB190114C consist of 4 OBJECT files, 22 BIAS files, and 12 FLAT files. The four OBJECT images are taken in four different filters. The filters used are the g, r, i and z. The imaging started at 21:20:56 14th of January 2019, the same day as the event. This result in the first image, taken in the g-filter, was imaged with an average time of exposure at only 26 minutes after BAT was triggered.

3.2.4 GRB190829A

The data set from GRB190829A consist of 22 OBJECT files, 22 BIAS files, and 26 FLAT files. The OBJECT files consist of six images in the r-filter, five in the u-filter, five in the z-filter, three in the i-filter, and three in the g-filter. The first three images taken are in the r-filter with exposure times of 300 seconds. The exposure times for the rest of the images are 60 seconds. After the first three images in the r-filter are taken, the images are taken in the u-filter which is followed by the imaging in the z-filter. After the imaging in the z-filter, the following three images are done in the r-filter. After this, the i-filter is used followed by the g-filter. The imaging started at 01:58:25 30th of August 2019, the day after the event triggered BAT. The first image was taken 364 minutes after BAT got triggered, corresponding to a little later than 6 hours. The last image was taken 7.7 hours after triggering BAT.

3.2.5 GRB191019A

The data set from GRB191019A consist of 14 OBJECT files and 15 FLAT files. No BIAS files were in the data set. Imaging is done in the i, g, r, z filter. Imaging began at 19:43:39 19th of October 2019, the same day as the event. The exposure times of the first nine images are of 300 seconds in length.

The last five images have exposure times of 200 seconds. The average time of exposure of the first image is 273.4 minutes after the event triggered BAT.

3.2.6 GRB211211A

The data set from GRB211211A consist of 6 OBJECT files, 22 BIAS files, and 9 FLAT files. The OBJECT files consist of two images in the r-filter, two images in the g-filter, and two images in the i-filter. All the images have exposure times of 120 seconds. The imaging started at 05:44:42 12th of December 2021, the day after the event. The average time of exposure of the first image taken is 996 minutes after BAT got triggered.

3.2.7 GRB220101A

The data set from GRB220101A consist of 3 OBJECT files, 10 BIAS files, and 3 FLAT files. The three OBJECT files are all images taken in the r-filter with exposure times of 120 seconds. The first image has an average time of exposure of 945 minutes after BAT got triggered. Moreover, the start of exposure for the first image began at 20:54:28 1st of January 2022, the same day as the event.

3.2.8 GRB220521A

The data set from GRB220521 consist of 16 OBJECT files, 32 BIAS files, and 9 FLAT files. Of the OBJECT files four of the images are taken in the r-filter whereas the remaining 12 images are taken in the z-filter. In this report, only the four images in the r-filter were used. Imaging began at 23:35:14 21st of May 2022, the same day as the event. The first, second, and fourth image in the r-filter has exposure times of 300 seconds each. For the third image, an exposure time of 218 seconds is used. The average time of exposure for the first event is taken 17.2 minutes after BAT got triggered.

4

RESULTS

4.1 Usability of PyNOT

When it comes to PyNOTs usability, I have managed to run the pipeline both through a terminal and scripts. To test the functionality of PyNOT, I have tested PyNOT on seven different GRB events. However, only for five of the seven data sets was this possible. The GRB events I have tested PyNOT on are listed in table 1 where it is also shown that PyNOT could not be run with the data set from GRB190829A nor the data set from GRB191019A. For the data set of GRB190829A, PyNOT raised a value error that PyNOT was not able to handle. For the data set of GRB190119A, the lack of BIAS files is a problem as PyNOT needs at least three BIAS files to work.

Event:	Did it run?	Time for "Initialize"	Time for "run_pipeline"	Time for "find_new_sources"	Total run time
GRB190114A	Yes	0.63s	15.77s	1.50s	17.92s
GRB190114B	Yes	0.82s	63.09s	3.93s	67.85s
GRB190114C	Yes	0.71s	23.66s	2.99s	27.37s
GRB190829A	No	...	...	...	...
GRB191019A	No	...	...	...	...
GRB211211A	Yes	0.66s	31.85s	2.90s	35.42s
GRB220101A	Yes	0.19s	16.00s	0.87s	17.07s

Table 1: *This table shows the GRB events of which PyNOT was tested on and their respectively run times. The table reveals that PyNOT did not work for GRB190829A and GRB191019A. It can also be seen that most of the time is spent in the "run_pipeline" function, which is the function doing the data processing of the raw images and object detection.*

Besides the fact that PyNOT has the capability of doing data processing, table 2 shows, how well it does when it comes to detection of transients in the reduced image. The magnitude threshold in PyNOT is by default 20.1. However, this threshold is changeable with an input parameter. As it can be seen in table 2, I have tried with two different apparent magnitudes. I have done this, as it shows the effect it can have if foreknown knowledge of the apparent magnitude of a GRB event is available. I have tried with the default threshold and a slightly lower threshold of 20.0. This was to observe how many sources PyNOT would sort away if a tiny difference is made. This can be important as too many sources are not necessarily a good thing. It revealed, that one to three sources were sorted out depending on the event. In the cases where the transient was found at the default magnitude threshold of 20.1, the transient was still present when the magnitude threshold was lowered to 20.

As it is shown in table 1, PyNOT was capable of successfully identifying the transient in GRB190114C and GRB220101A. In both cases, an unidentified source was consistent with both BAT and XRT error circles and in both cases, this was the actual transient. The transients are marked with green by

PyNOT as shown in figure 5 and 13. In those cases PyNOT does find the transient, it finds nine other candidates in the field of GRB190114C and seven in the field of GRB220101A. For GRB190114C, three of the sources were found in the r-filter, where one of them is the GRB (Figure 13). For the three other events, PyNOT found several candidates for GRB190114B and GRB211211A. However, none of them were the actual transient. Looking at GRB190114A no candidates were found at all.

The magnitude threshold was raised to 21.5 for GRB190114A, GRB190114B, and GRB211211A to see if the transients then would be detected. The results for the r-filter are shown in figure 35, 36 and 37 together with the results from when the magnitude threshold was at the default 20.1. The images in the right side of figure 35, 36 and 37 are the resulting images when the magnitude threshold was increased to 21.5. In those images, which is when the threshold is raised to 21.5, it is clear that PyNOT managed to find the transients in the fields of GRB190114A, GRB190114B, and GRB211211A.

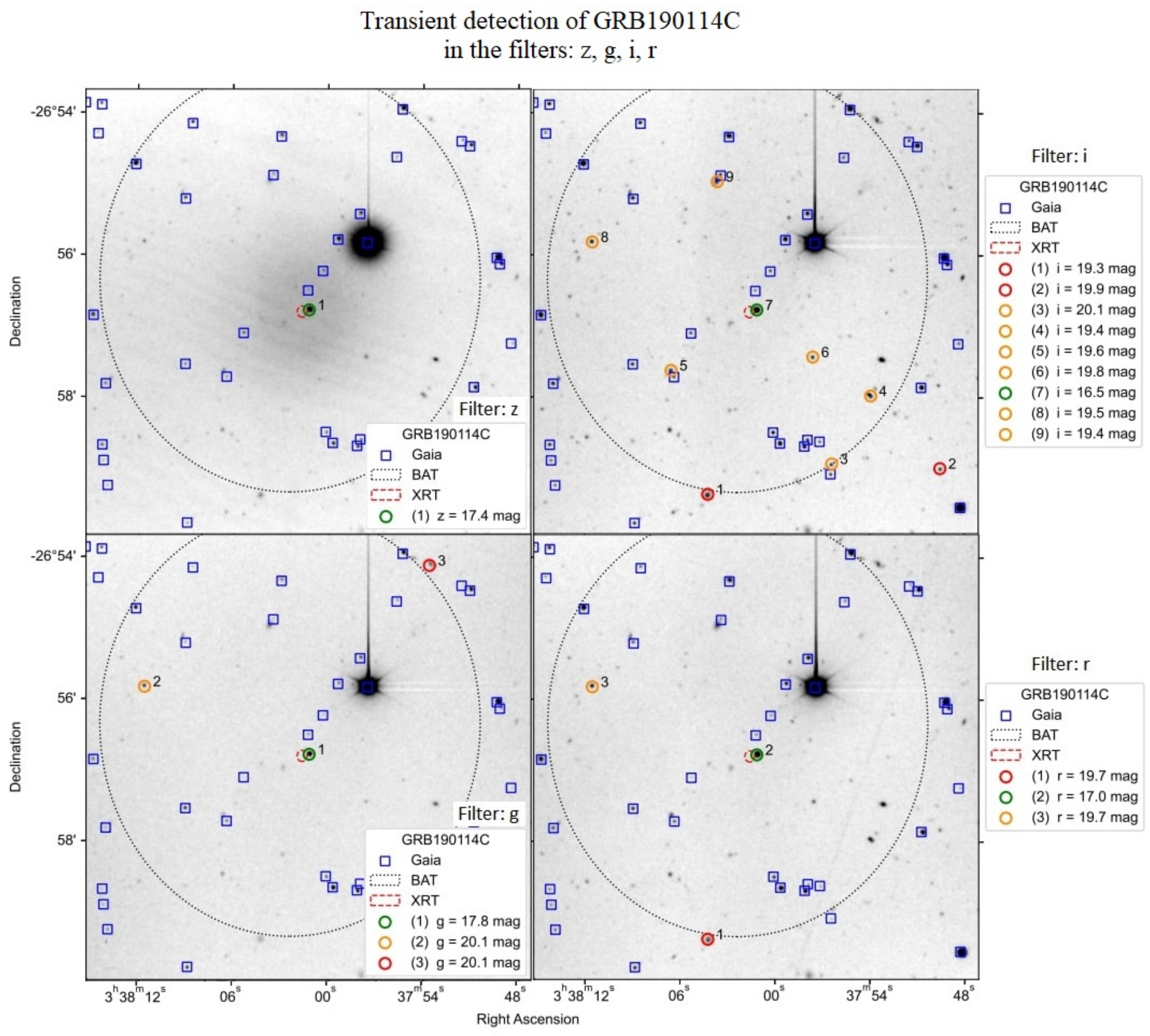


Figure 13: *This figure shows the images from transient detection of GRB190114C in the g, r, i, and z filters. A source consistent with both BAT and XRT is found and marked with green. The source marked with green is the transient. It can also be seen that the number of transient candidates varies between filters, yet the brightest source in all four filters is the transient.*

PyNOTs runtime has also been tested (Table 1). Here, it should be noted that a small modification is added to run PyNOT in a script in a Jupyter Notebook. However, the modification only affects the

Event:	Transient candidates. mag = 20.0	Transient found? mag = 20.0	Transient candidates. mag = 20.1	Transient found? mag = 20.1	Filters
GRB190114A	0	No	0	No	r
GRB190114B	15	No	18	No	g, i, r, z
GRB190114C	8	Yes	10	Yes	g, i, r, z
GRB211211A	6	No	7	No	g, i, r
GRB220101A	6	Yes	8	Yes	r

Table 2: *The total number of transient candidates found in all filters available for the corresponding GRB event is shown in this table. The results are with the magnitude threshold used for comparison with the Gaia catalog at 20.0 and 20.1. In only two of the five events, the actual GRB was detected. Further, it can be seen that in the data from GRB190114A, no transient candidates were detected using these thresholds. It can also be seen that the slight change in the threshold lowered the number of candidates for all the events.*

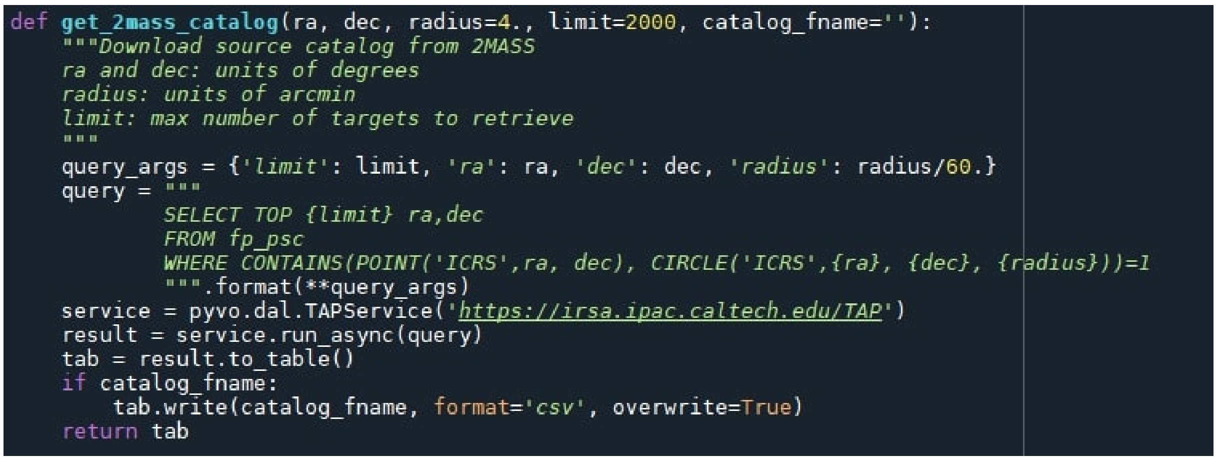
total runtime and the effect was found to be around 0.01 seconds. Nonetheless, the runtimes for the functions are for all the available filters for the given event. In table 1, it is clear that most of the time is spent on data processing of the images done by the *"run_pipeline"* function. It is also clear that the total run time is varying depending on the event. This is found to be correlated with the number of sources present in the field of observation, the number of raw files to be processed, and the number of filters for transient detection. More sources, files, and filters lead to longer run times. Of those data sets PyNOT did finish, PyNOT had for each of those events a total runtime of less than a minute, with GRB190114B being an exception. For GRB190114B the total run time was 67.85 seconds.

4.2 Extension of PyNOT

4.2.1 Documentation and changing the source code of PyNOT

For further extension of PyNOT with the objective to improve PyNOTs versatility and robustness, several changes to the source code have been made. The changes have been made for different reasons. Some, as it might be a useful feature to have inside PyNOT itself, like making PyNOT use several catalogs. And some, to save data in a handy way for later use outside PyNOT, but as an adjunct to it, like for point source selection. The changes inside PyNOT for using more catalogs have been done in both WCS.py and the transients.py source code files. In the WCS.py file, the library *"pyvo"* is imported and the function *"get_2mass_catalog"* is added. The *"get_2mass_catalog"* function is showed in figure 14. A function named *"get_wise_catalog"* which is almost identical to the one in figure 14 is added right after in the script, but is not shown here. The difference between the two functions added is the name of the catalog, which is deciding which catalog to fetch. It can be seen that in the line *"query"* there is a parameter *"FROM"* which has the argument *"fp_psc"*. *"fp_psc"* is the name of the 2MASS catalog. For the wise function, this has been changed to *"allwise_p3as_psd"* as this is the name of the WISE catalog used. These functions query the catalog 2MASS and WISE catalog respectively when requested.

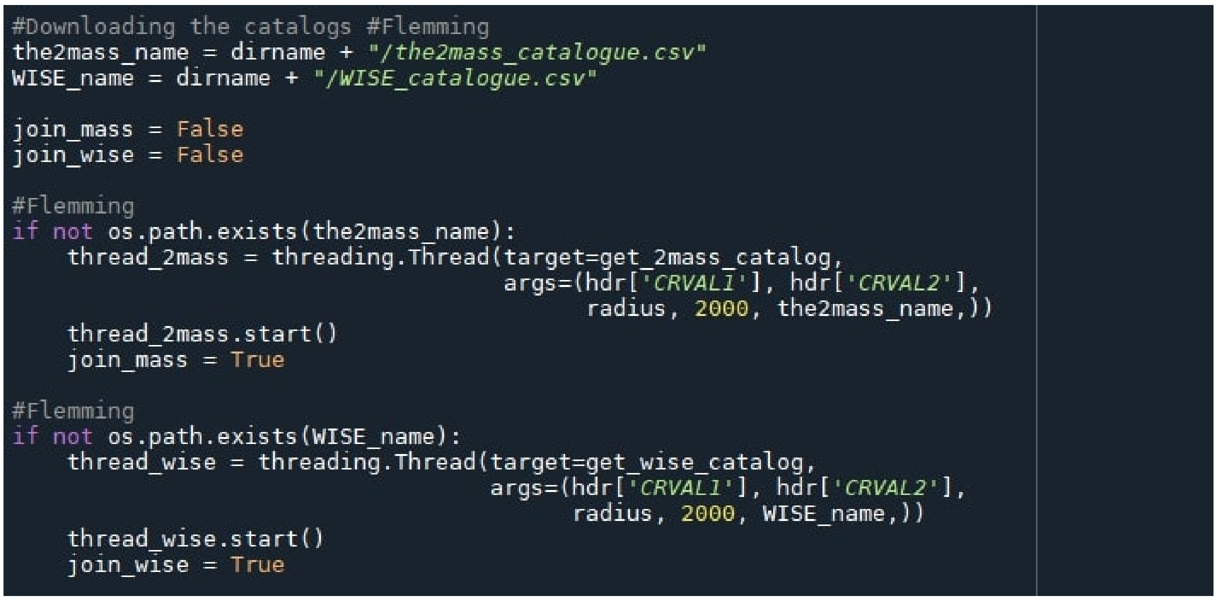
For it to work efficiently I have also changed the code, such that it uses threading as explained in methodology (Figure 12). This was done in the *"correct_wcs"* function in the WCS.py source code



```
def get_2mass_catalog(ra, dec, radius=4., limit=2000, catalog_fname=''):
    """Download source catalog from 2MASS
    ra and dec: units of degrees
    radius: units of arcmin
    limit: max number of targets to retrieve
    """
    query_args = {'limit': limit, 'ra': ra, 'dec': dec, 'radius': radius/60.}
    query = """
            SELECT TOP {limit} ra,dec
            FROM fp_psc
            WHERE CONTAINS(POINT('ICRS',ra, dec), CIRCLE('ICRS',{ra}, {dec}, {radius}))=1
            """.format(**query_args)
    service = pyvo.dal.TAPService('https://irsa.ipac.caltech.edu/TAP')
    result = service.run_async(query)
    tab = result.to_table()
    if catalog_fname:
        tab.write(catalog_fname, format='csv', overwrite=True)
    return tab
```

Figure 14: *This image shows the function that is added in PyNOT to query the 2MASS catalog. An almost identical function is added for WISE as well. The catalogue fetched with this function, is the "fp_psc" as it can be seen next to "FROM". The "fp_psc" is the name of the 2MASS catalog. The function also saves the data fetched if a catalog name is given as input to the function, which can be seen in the third line from the bottom.*

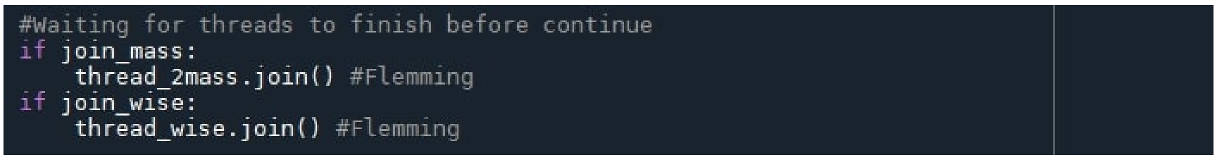
file as well. I started off by importing the library *"threading"* at the top of the document. Further down in the source code inside the function *"correct_wcs"*, the code shown in figure 15 was added. This checks if the catalog already exists. If it does not exist, a new thread will be spawned that calls the function that queries the missing catalogs. This is done for both 2MASS and WISE. It can also be seen that the variables *"join_mass"* or *"join_wise"* will be set to *"True"* if the catalog is not initially found. To join the threads again, the code has been changed further down at the end of the same function. The code in figure 16 has been added, and if the above variables are set to *"True"*, the script will wait for the threads to finish, ensuring everything is done before continuing.



```
#Downloading the catalogs #Flemming
the2mass_name = dirname + "/the2mass_catalogue.csv"
WISE_name = dirname + "/WISE_catalogue.csv"

join_mass = False
join_wise = False

#Flemming
if not os.path.exists(the2mass_name):
    thread_2mass = threading.Thread(target=get_2mass_catalog,
                                    args=(hdr['CRVAL1'], hdr['CRVAL2'],
                                          radius, 2000, the2mass_name,))

    thread_2mass.start()
    join_mass = True

#Flemming
if not os.path.exists(WISE_name):
    thread_wise = threading.Thread(target=get_wise_catalog,
                                   args=(hdr['CRVAL1'], hdr['CRVAL2'],
                                         radius, 2000, WISE_name,))

    thread_wise.start()
    join_wise = True
```

Figure 15: *This section of code is added into PyNOT. It checks if threads should be spawned and if so, spawning threads to query the catalogs. As indicated in the code, two separate threads can be spawned for 2MASS and WISE. The spawned threads will call the function set as "target" with the parameters given to the "args" argument.*

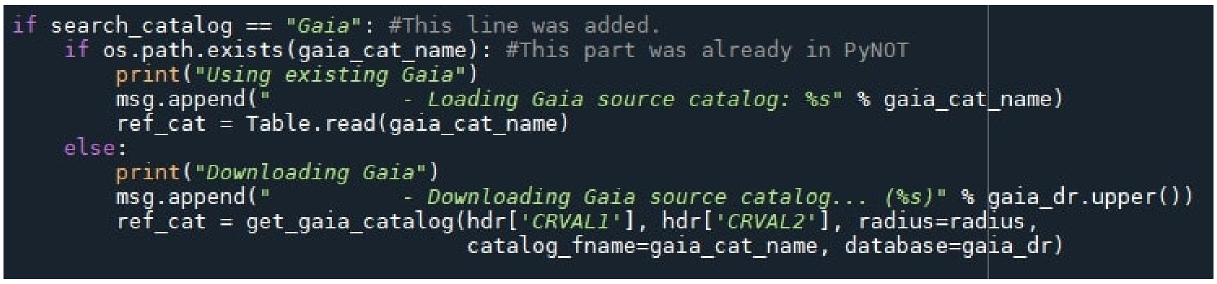
In the transient.py source code file, where the WISE and 2MASS catalog is used, changes have been made in the *"find_new_sources"* function such that it takes an additional input parameter. The new



```
#Waiting for threads to finish before continue
if join_mass:
    thread_2mass.join() #Flemming
if join_wise:
    thread_wise.join() #Flemming
```

Figure 16: *This code snippet is added code to the PyNOT source code. This ensures if threads are spawned to query 2MASS or WISE, that the threads have finished before continuing. This guarantees the catalogs are fetched before starting new tasks and an error later in the PyNOT pipeline will not occur.*

input is named *"search_catalog"* and takes in the name of the catalog wished to be used for transient detection and thereby coordinate comparison. It has been done in this way, so you do not have to use transient detection with all three catalogs if one wants to save time. The three options for input are *"Gaia"*, *"2Mass"*, and *"WISE"*, which clearly indicate which catalog is used in the transient detection. This part of the code is changed with straightforward if-statements and minor changes to the part in figure 17, which already were in PyNOT. Right under this snippet of code is a very similar snippet, but instead of *"if"*, *"elif"* is used. The *"elif"* statement is followed by another argument than *"Gaia"* and then very similar code but for 2MASS instead. Then another snippet for WISE as well. This could be reduced from three pieces of code into one. However, I have chosen to have three snippets of code as it is more clear what is happening. But, if an arbitrary number of catalogs should be used, I would modify this part to be more versatile.



```
if search_catalog == "Gaia": #This line was added.
    if os.path.exists(gaia_cat_name): #This part was already in PyNOT
        print("Using existing Gaia")
        msg.append("          - Loading Gaia source catalog: %s" % gaia_cat_name)
        ref_cat = Table.read(gaia_cat_name)
    else:
        print("Downloading Gaia")
        msg.append("          - Downloading Gaia source catalog... (%s)" % gaia_dr.upper())
        ref_cat = get_gaia_catalog(hdr['CRVAL1'], hdr['CRVAL2'], radius=radius,
                                   catalog_fname=gaia_cat_name, database=gaia_dr)
```

Figure 17: *This is a part of modified code in PyNOTs "find_new_sources" function for transient detection. It shows how PyNOT now can choose between catalogs to use by using an input argument. The rest of the code was already in PyNOT but is also shown as it is part of the context. In this snippet, the input is "Gaia". Similar snippets are added for 2MASS and WISE with "elif"-statements instead of an "if"-statement.*

I also made PyNOT save a PICKLE file with the use of the library *"pickle"*. It was the best way I could figure out to save data for making an image with some changes, but still very close to the original one made during transient detection. This was changed as it is used in the point source selection pipeline. The changes have been done at the end of the *"find_new_sources"* function with the code shown in figure 18. A dictionary of the data is created and filled with the data used for making the plot in transient detection. The data is then saved, such that recreating the image later is very easy by importing the data in another script with the use of *"pickle"*. After importing the data, it can be edited and then plotted again by reusing the code used for plotting in the source code of PyNOT.

Furthermore, I changed PyNOT to also save a PNG file of the images for transient detection, whereas PyNOT before only saved a PDF file. I will not show the code here as it is a copy of already existing code with ".png" instead of ".pdf" found in one of the lines at the end of the transient.py source code file.

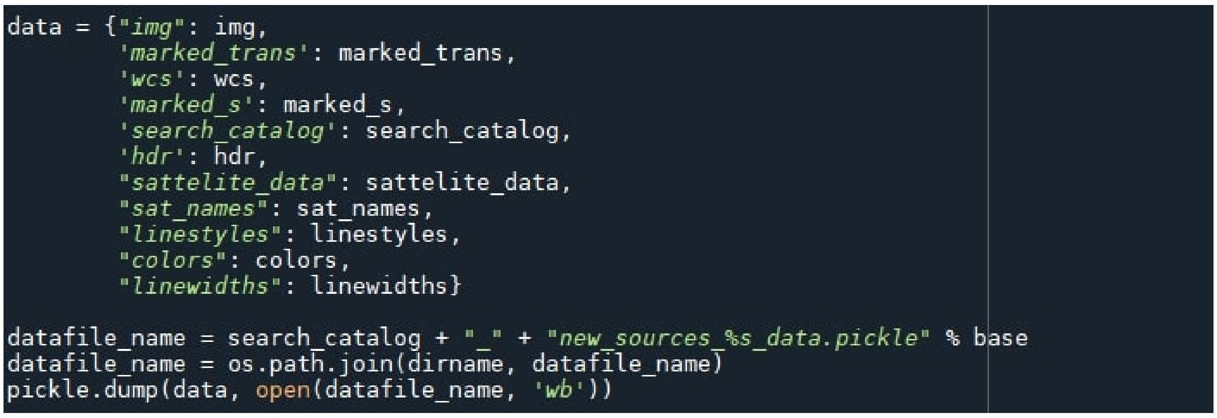

```
data = {"img": img,
        'marked_trans': marked_trans,
        'wcs': wcs,
        'marked_s': marked_s,
        'search_catalog': search_catalog,
        'hdr': hdr,
        "sattelite_data": sattelite_data,
        "sat_names": sat_names,
        "linestyles": linestyles,
        "colors": colors,
        "linewidths": linewidths}

datafile_name = search_catalog + "_" + "new_sources_%s_data.pickle" % base
datafile_name = os.path.join(dirname, datafile_name)
pickle.dump(data, open(datafile_name, 'wb'))
```

Figure 18: *This is code added into PyNOT that gathers the data used when creating the image PyNOT makes with the "find_new_sources" function for transient detection. The data is gathered in a dictionary and then saved into a PICKLE file. This is becoming useful when some want to make slight modifications to the plot created by the "find_new_sources" function, which is the case for the point source selection.*

4.2.2 Make PyNOT use several catalogs

I managed to extend PyNOTs capability by comparing the sources detected with other catalogs than Gaia. I made it work so PyNOT also can use the 2MASS and WISE catalogs by changing the code as explained above. Table 8 shows that it has a quite huge impact on the execution time. Extending PyNOT showed an increase in the average total runtime of 93.58 percent, which is resulting in an average total run time of 64.12 seconds. No significant difference in runtime was found in the *"run_pipeline"* function. However, transient detection with the Gaia catalog showed an increase in runtime of 72.52 percent.

Event:	”run_pipeline”	Gaia run time	2MASS run time	WISE run time	Total run time
GRB190114A	16.13s	2.28s	4.07s	6.78s	29.99s
GRB190114B	58.72s	6.59s	16.42s	19.88s	102.61s
GRB190114C	21.62s	5.74s	21.90s	19.01s	69.06s
GRB211211A	39.09s	4.83s	35.30s	11.75s	91.89
GRB220101A	15.92s	1.58s	4.88s	4.48s	27.07s
Average times	30.30s	4.21s	16.51s	12.38s	64.12s

Table 3: *Table showing run times of PyNOT after being extended to use several catalogs. The runtime of the ”run_pipeline”-function remains almost unchanged. The changes done in the "find_new_sources" function seem to affect both the run time of transient detection using Gaia, as well as the total run time is increased due to transient detection being done in more filters.*

Further, it can be seen in table 4 that PyNOT found more potential candidates across the different filters when using the 2MASS catalog compared against the Gaia catalog. When using the WISE catalog, the number of potential transients was found to be around the same as using the Gaia catalog but varying for each event individually. As table 4 shows, it was found that, when using the WISE catalog, PyNOT did not manage to find the actual transient, indicating that the coordinates of the transient already being in the catalog as a different object. While using the 2MASS catalog, the transient was found in the same events as using the Gaia catalog for comparison.

Event:	2MASS Number of unknown sources. Mag = 20.1	2MASS Was transient found?	WISE Number of unknown sources. Mag = 20.1	WISE Was transient found?
GRB190114A	10	No	7	No
GRB190114B	28	No	9	No
GRB190114C	23	Yes	9	No
GRB211211A	16	No	6	No
GRB220101A	22	Yes	12	No

Table 4: *This table shows the number of transient candidate sources found across filters available using 2MASS and WISE instead of Gaia. It also shows if the transient was found using those catalogs.*

4.2.3 Point source selection

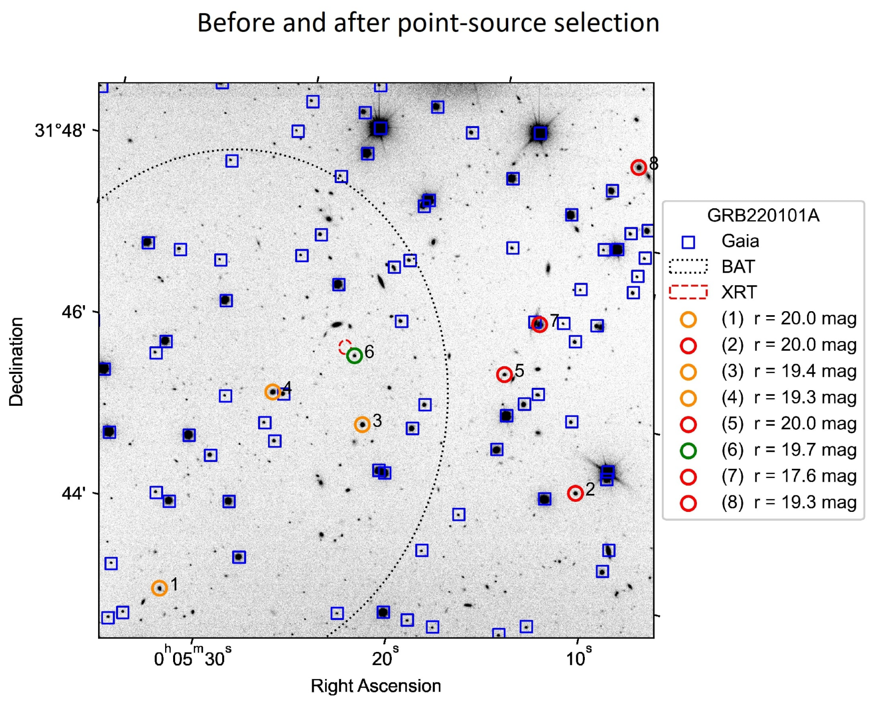
Sorting out those sources which were not point sources was also done. This was done by comparing the FWHM of individual sources against the FWHM distribution of all the sources. The images taken in the r-filter are shown in Appendix 2. The results of the image from GRB220101A compared with the Gaia catalog in the r-filter are shown in figure 19. Figure 19 shows that by filtering out those sources which most likely are not point sources in the image of GRB220101A, we are left with only one out of the eight potential candidates. This one candidate is also the actual transient.

An overview of the takeaways of all the results of the point source selection pipeline is presented in table 5 and 6. In table 5 the results are from GRB190114 and GRB220101A as those are with the transient identified by PyNOT. Table 6 contains the results of the point source pipeline applied to the remaining events where the transient was not detected with the default magnitude threshold. In table 5, it is revealed that using the Gaia catalog and point source selection in the image of GRB190114C, only one source is left across all the filters, which is the transient. Hence, for both cases where the transient was found by PyNOT, only the transient is the one remaining when using the Gaia catalog and the r-filter. For both GRB190114C and GRB220101A, the transients were also found when using the 2MASS catalog. However, using the point source selection pipeline did not sort out that many sources, leaving the image still filled with distracting candidates. Looking across all the results for the events presented both in table 5 and 6 shows that some sources are sorted out almost every time. However, in every case where the transient was detected, the transient was kept.

The pipeline for point source selection should also count in the runtime. The point source selection pipeline has been tested and the results are presented in table 7. The test reveals an average runtime of 26.42 seconds if the point source selection pipeline is applied to all filters and using both Gaia, 2MASS, and the WISE catalog. If only using the Gaia catalog, the average run time is tested to be 13.52 seconds (Table 7).

4.2.4 Detecting fading objects

I have tried to figure out if it would be possible to detect the fading of transients with images taken from NOT. In the attempt to use fading to detect transients, the data from GRB220521A was used. A theoretically expected drop was calculated with equation 9 with an alpha value of 1.53, used from



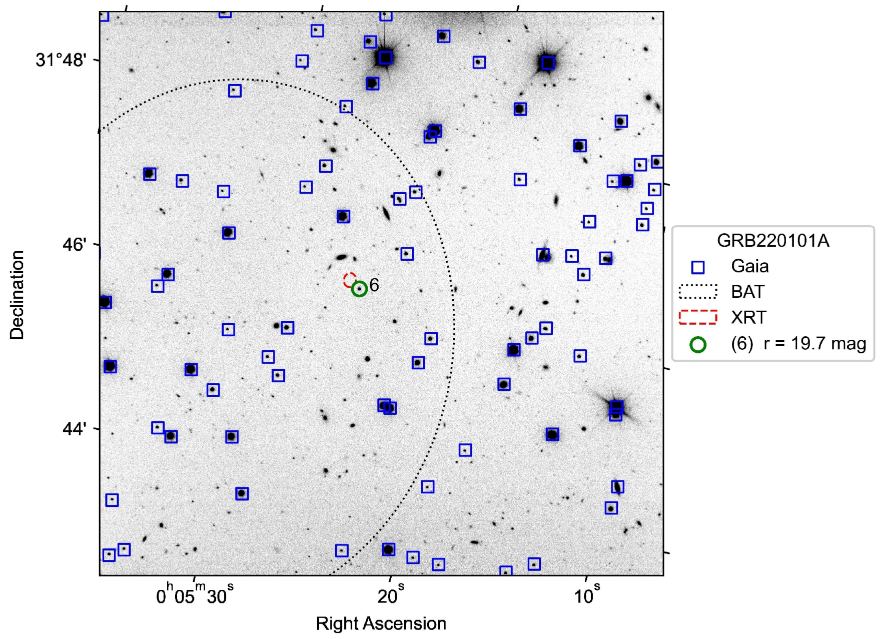


Figure 19: *These two images are from GRB220101A before and after point source selection. In the top image, PyNOT has done transient detection, while in the second image, the point source selection pipeline has been applied as well. It shows that the only source remaining after the point source selection pipeline has been applied is the one consistent with both BAT and XRT. The remaining sources are sorted out, as they were not found to be point sources.*

GCN [36]. The images used were taken 14.9, 20.3, and 25.8 minutes after first triggering by BAT at 23:20:21 UT [36]. The average times of exposure were calculated to be 17.4, 22.8, and 27.6 minutes after the trigger. This revealed a theoretical drop in magnitude at 0.4523 from the first to the second image and 0.3148 from the second to the third image, and thus 0.7672 from the first to the third image. This was used as a reference to compare the following results against.

Event, catalog, filter	Number of sources	Number of point sources	Is the transient kept
GRB190114C			
2MASS: g	6	4	Yes
2MASS: i	21	10	Yes
2MASS: r	11	6	Yes
2MASS: z	8	7	Yes
WISE: g	2	2	...
WISE: i	8	6	...
WISE: r	5	3	...
WISE: z	3	3	...
Gaia: g	3	1	Yes
Gaia: i	9	1	Yes
Gaia: r	3	1	Yes
Gaia: z	1	1	Yes
GRB220101A			
2MASS: r	22	14	Yes
WISE: r	12	8	...
Gaia: r	9	1	Yes

Table 5: *This table shows the number of sources before and after the point source selection pipeline has been applied to the images of GRB190114C and GRB220101A. If a transient was not found in the images initially it is marked with "..." in the column "Is the transient kept". From the table, it is clear that in no image with the transient detected, the point source selection pipeline has sorted it away. Moreover, it can be inferred from the table that using the Gaia catalog will only leave one source after the point source selection pipeline has been applied in the event of GRB190114C and GRB220101A.*

For the following, I separated the OBJECT-images from GRB220521A into different folders such each final image from PyNOT was created using only one initial OBJECT-image. I found that in the r-filter there were four OBJECT-images, but in only three of them, the transient where actually present. The three earliest images were for this reason the ones used and the last was excluded.

4.2.4.1 *PyNOTs Mag_Auto*

I have tried by using the PyNOTs "*Mag_Auto*" value that PyNOT calculates when doing object detection. I ran PyNOTs transient detection pipeline on all three images and got 938, 946, and 629 sources that are not found in the Gaia catalog. Cross-matching by coordinates was done between the images, leaving 26 sources that repeated in all three images. There were obviously plenty of other sources that repeated. However, cross-matching sorted many of those out due to the WCS calibration leading to coordinates varying slightly, and with that many sources, many were misidentified between the images. In a less crowded image, the uncertainty used for RA and DEC in the cross-matching could have been increased to get more sources. But it was found, doing so in this field, would lead to sources being wrongly cross-matched. The transient was actually sorted out as well in the cross-matching, but I forced it to be kept based on the coordinates from XRT. Meaning, I had a total of 27 sources left. The 27 sources can be seen in figure 20 to the left. The "*Mag_Auto*" value was taken for each of the 27 sources in each image. The median of each source was calculated and subtracted from the magnitude of the corresponding source in each individual image to get the spread seen in figure 20 to

Event, catalog, filter	Number of sources	Number of point sources
GRB190114A		
2MASS: r	10	9
Gaia: i	7	6
GRB190114B		
2MASS: g	7	6
2MASS: i	17	12
2MASS: r	15	11
2MASS: z	23	12
WISE: g	4	4
WISE: i	6	3
WISE: r	5	5
WISE:z	6	6
Gaia: g	1	0
Gaia: i	8	1
Gaia: r	6	0
Gaia: z	15	2
GRB211211A		
2MASS: g	2	1
2MASS: i	14	8
2MASS: r	2	1
WISE: i	5	4
Gaia: g	1	0
Gaia: i	6	0
Gaia: r	1	0

Table 6: *This table shows the number of sources before and after the point source selection pipeline has been applied to the images of GRB190114A, GRB190114B, and GRB211211A. In those images, the transient was not initially found with the default magnitude threshold. The table shows that in most of the images, some sources are sorted away.*

	GRB190114A	GRB190114B	GRB190114C	GRB211211A	GRB220101A	Avg
Time Gaia	4.71s	28.16s	12.49s	12.33s	9.90s	13.52s
Time All	10.27s	54.32s	26.36s	26.34s	14.82s	26.42s

Table 7: *This table shows the run times for the point source selection pipeline. It has been applied on the Gaia images alone and it has been applied to all the images. The average run time for these five events is 13.52 and 26.42 seconds for only the Gaia image and all images respectively.*

the right. Here, it appears that the transient is fading. As all 26 sources were brighter than the transient and none of them had sudden changed in magnitude between images, all of them were kept. The 26 sources were used to establish a reference by taking the average of them in each individual image. This average in each image was subtracted from the transient in the respective image, leaving us with three values, one for each image. The drop in magnitude of the transient is then the difference between these values. The difference was taken between the first and the second, the second and the third, and the first and the third image. The respective errors were calculated with the use of error propagation to see if the drop was significant between images. This showed a drop of 0.076 ± 0.074 from the first to the second image and 0.125 ± 0.090 from the second to the last image and 0.201 ± 0.089 from the first to the last image. The drop between the first and the last image does not follow the theoretical drop in the magnitude of 0.7672. However, the drop in magnitude is significant as the detected drop is much larger than the error.

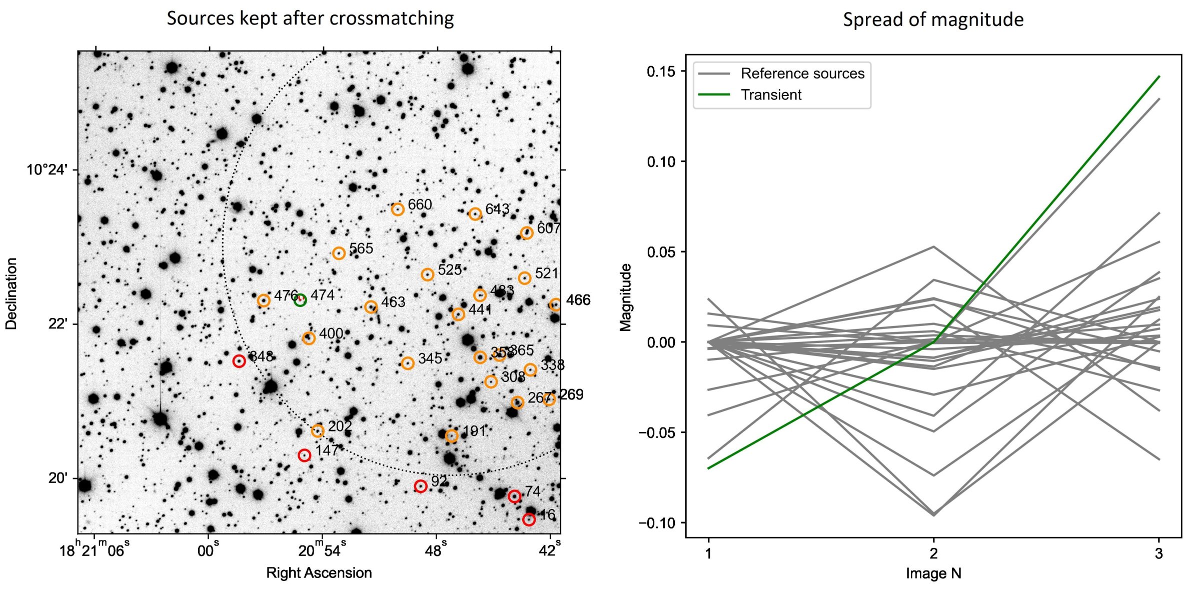


Figure 20: *The figure on the left shows, the sources repeating in all three images after cross-matching in the event of GRB220521A. The spread in magnitude of the sources between images is shown in the figure to the right. The spread in this figure is from the method using PyNOTs "Mag_auto".*

4.2.4.2 *Reusing Apertures*

Due to the cross-matching being difficult, as a result of the sources laying close, another approach was attempted. After the image reduction by PyNOT, the three images were cut into smaller images with the XRT coordinates in the center and using the WCS-calibrated information from PyNOT. This was followed by making object detection and creating apertures in the first image of the three. The apertures were then reused in the two following images, making cross-matching almost redundant as the pixel coordinates are the same in each image as they are bound to the apertures. Applying these apertures in all three images and calculating the magnitudes results in a slight shift, which gives a variation in the magnitude of sources between images. I removed those sources with a magnitude fainter than 23 as the magnitude of the transient was around 23. This gave a slightly different number of kept sources in each image. It therefore still had to be cross-matched, but as all the coordinates are bound to the pixel values most sources were kept. Further, sources already in Gaia as well as those which were stacked very close to each other were removed as well. Moreover, if a source was too oval, meaning the minor axis divided by the major axis ratio was less than 0.8, it was also sorted out. It resulted in having 346 sources left repeating in each image.

The method described of using *"Mag_Auto"* was then applied. As seen in the figure 21 of the spread, it looks like there is a systematic error. It also appears that one source is acting differently as it does not follow this pattern at all. It was for that reason identified and removed. In this approach, the baseline was also set by the average of the sources, not including the transient, in each individual image before subtracting it from the magnitude of the transient, just like before. The differences between the resulting magnitudes were calculated and the drop was found to be 0.079±0.135 and 0.094±0.151 from the first to the second image and from the second to the third image, respectively. From the first to the third image, the drop was found to be 0.173±0.150.

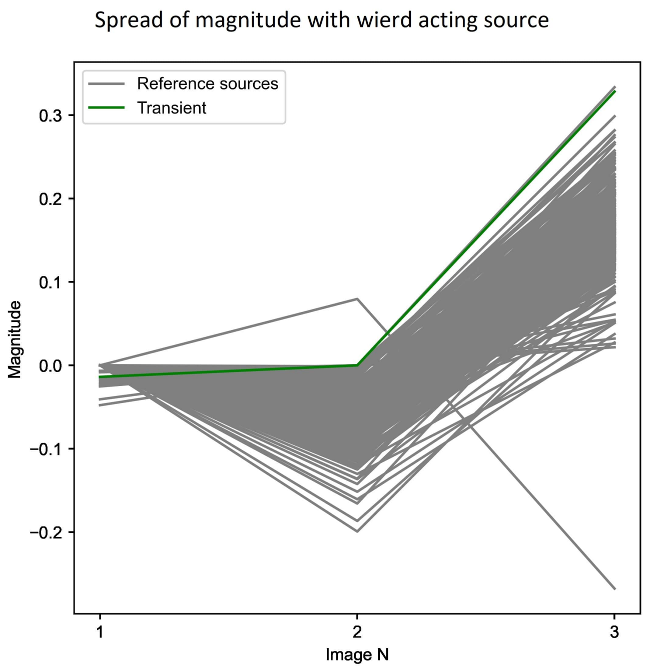


Figure 21: *Here, the spread in magnitude of the sources between images is shown. The spread is from the method using fixed apertures in all images. The figure shows a source that is not following the systematic error between the images. This source was identified and sorted out.*

4.2.4.3 *Point spread function*

In the pursuit of getting the error down, another approach was tested. Using Photutils to use a PSF, I got the error a bit down. For the method of selecting sources, I started off by using the latter of the two above methods described. This resulted in drops of -0.019±0.016 and 0.044±0.015 between the first and the second image and between the second and the third image, respectively. This revealed an increase in intensity from the first to the second image even when uncertainties are considered. From the first to the third image, the drop was 0.025±0.013. The error was lowered. However, the detected drop was also reduced.

Lastly, I tried applying a PSF together with the method of using cross-matching between images created with PyNOT, as the first method described. The spread between the images resulted in figure 22. Calculating the drops between images resulted in a drop of 0.060±0.076 from the first to the second image and 0.182±0.035 from the second to the third image. From the first to the third image, a drop was found to be 0.243±0.072.

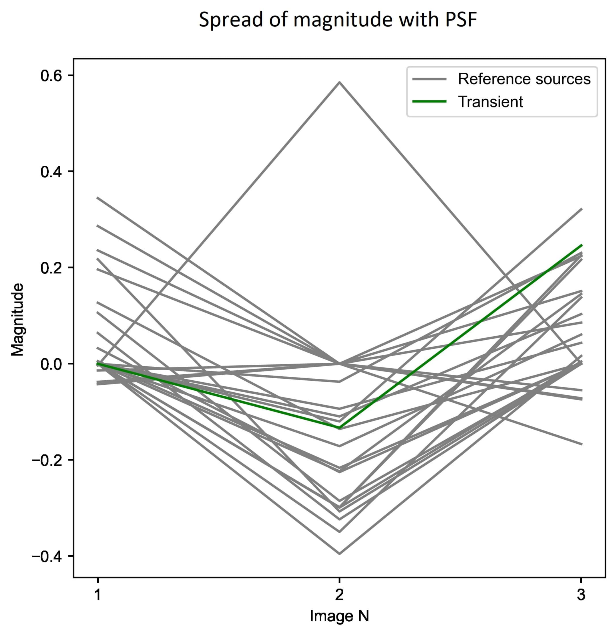


Figure 22: *Here, the spread in magnitude of the sources between images is shown. The spread is from the method of using cross-matching of sources and PSF for finding magnitudes.*

4.3 Test of PyNOTs system using coordinates for transient identification.

A test of how PyNOTs system of using coordinates for identification was carried out to see how robust it is. The test showed how often a source would be too close to an already known object if placed randomly in a certain area of the sky. The test was executed by generating 20000 random points and observe how many matches it would get. It is considered a match when a random point is within an angular distance of 1.5 arcseconds of an object in a catalog. The angular distance of 1.5 arcseconds was chosen as this is the distance PyNOT use. I did the simulation 50 times for each area and took the mean, from where I converted it to a percentage of how often a match will occur. This test was done at a span of galactic heights ranging from -89 degrees to 89 degrees with right ascension held at zero in the frame of the galactic coordinate system. It was then repeated with galactic height held at zero and the right ascension varied between 0-360 degrees. For both tests, the area was changed by one degree for each run.

I queried the three different catalogs Gaia, 2MASS and WISE and made the tests for all three. As it can be seen in figure 24, 26 and 25 where the test is presented as a function of galactic height, the catalogs vary in number of sources. From figure 24, 26 and 25, it is clear that there is a direct correlation between the density of objects in an area and the probability that there is a match. The same pattern is seen when it is a function of right ascension as in figure 27, 28 and 29. In both cases, the density and so the probability of an unwanted match, is highest towards the center of our galaxy as expected.

PyNOT uses a distance of 1.5 arcseconds as a threshold for whether there is a match or not. Therefore, I have tested how much this threshold has an effect on giving a match. I have tested this in the area of GRB220101A. Again, I have produced random points in the area of which a cone search has been

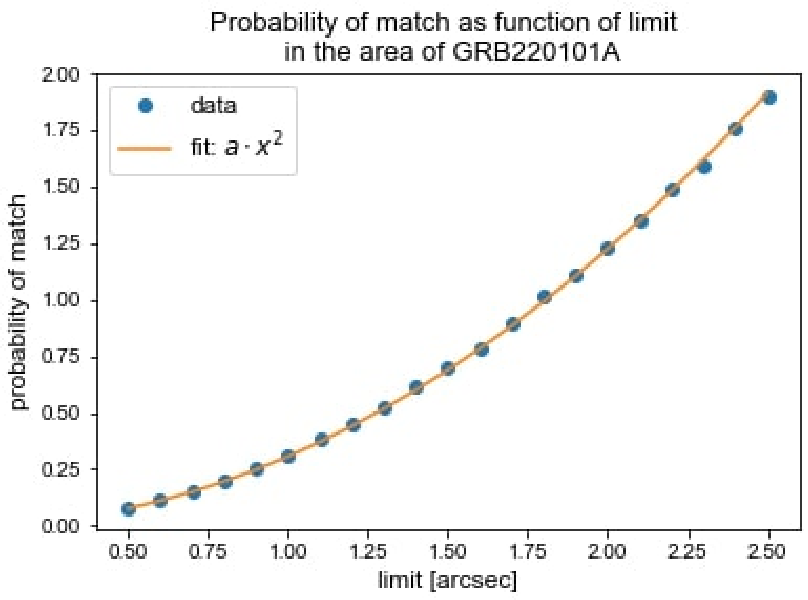


Figure 23: *This figure shows the result of simulating the effect of changing the angular distance threshold in the area of GRB220101A. For the simulation, random sources were generated. It is shown that a quadratic function is fitted to the data. This indicates that the probability of a match as a function of the angular distance threshold follows a quadratic function.*

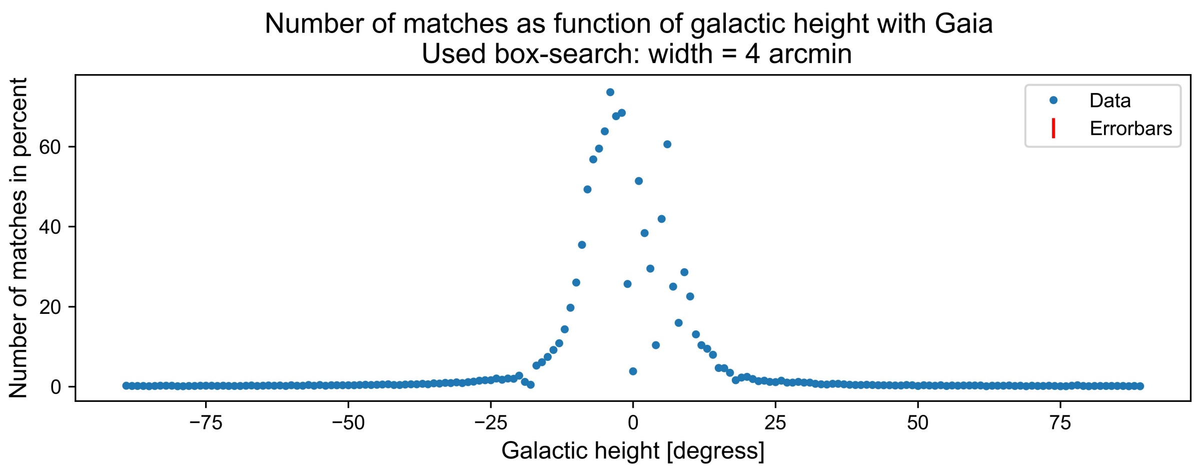
made and used the angular distance to judge if it is within the distance of an already known object. As figure 23 shows, the probability of getting a match follows a quadratic function just like the area of a circle, which in practice is also what is changed when the threshold is increased or decreased.

4.4 Graphical User Interface

I managed to create a GUI for the PyNOT pipeline together with my own pipelines for point source selection including cross-matching, selecting point sources, recreating plots and sorting images for easier access in folders. The GUI ended up being split into four layers, where three of them worked as wished. Why one of them did not and the importance of this will be discussed in the discussion.

4.4.1 Frontpage

The first layer presented in figure 30, is the front window that shows up when you initialize the GUI. This layer consists of the most necessary options, which the user has to manually act on. As seen in figure 30, the options are quite limited. The inputs have been put in the order of priority. First, you can choose if you have already run the PyNOT pipeline on your own. In this way, you will only be presented to the data, and do not have to run the whole pipeline again, which saves some time. Therefore, this checkbox is more for reviewing data already processed before. However, before continuing, you must select the folder in which you have the folder containing the raw files. The name of the folder is used for labeling some figures. Therefore, a good name for the folder could be the name of the GRB event. Once you have chosen your folder, the path will be displayed beside the label $''Your folder:''$.



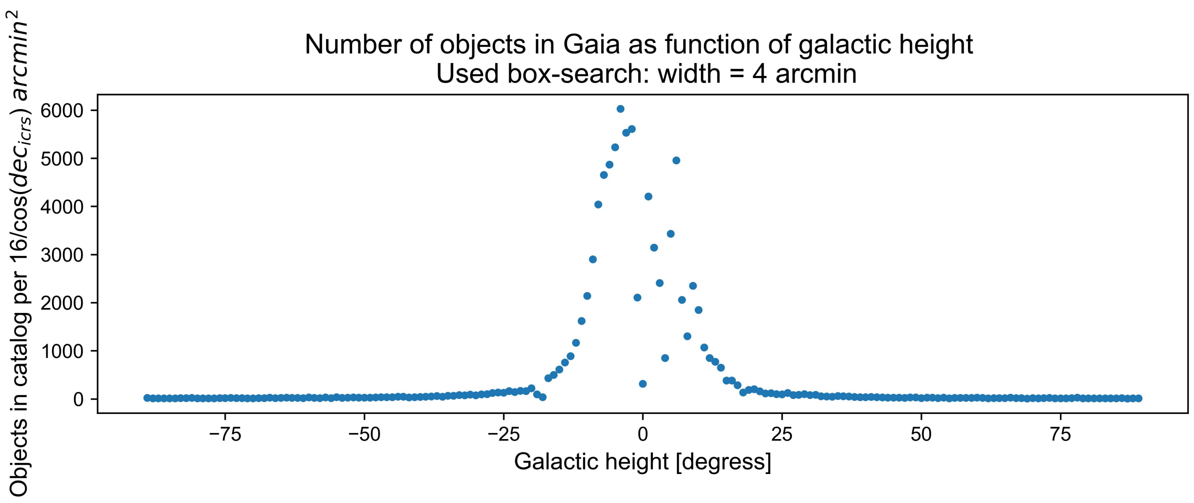
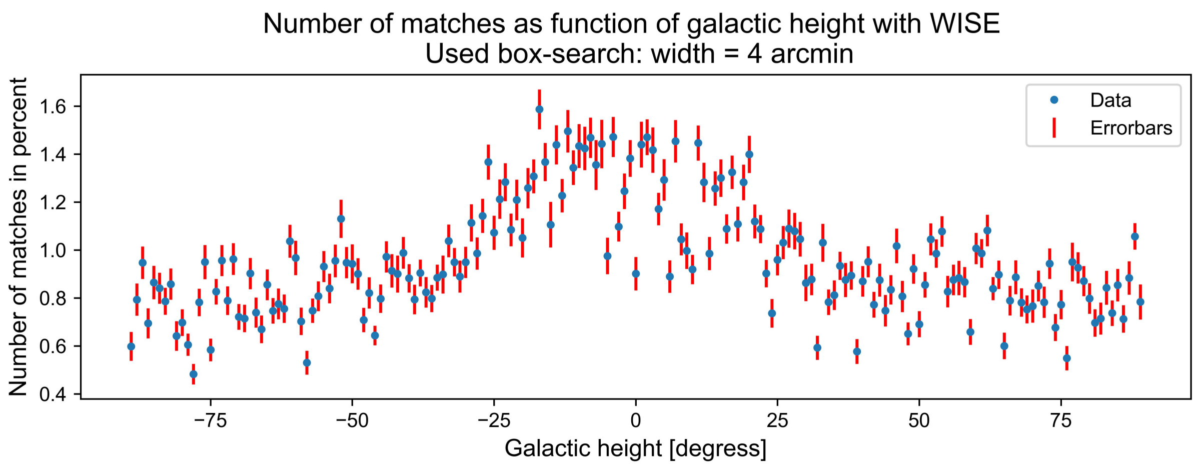


Figure 24: *The figure at the top shows the probability there will be an unwanted match between a transient placed at random and a source in the Gaia catalog as a function of galactic height. It is considered a match if the randomly placed source is within a radius of 1.5 arcseconds of a source in the catalog. The second figure shows the respective number of sources in the catalog. The figures show there is a clear correlation between the density of sources and an unwanted match. Each data point at the top is the mean from simulating 20.000 random transients 50 times. The errors are the standard deviation of the mean. It can be seen around zero degrees, that something is going on. This might be due to dust. It is also worth noticing that the y-axis in the second figure is a function of DEC in ICRS, found if transforming between the coordinate systems due to how the catalog is queried.*

If you have checked the button *"Already ran PyNOT on the raw images"*, and chosen the correct folder, you can simply click *"Submit"*, and the GUI will take you to the *"Basicinformation"* layer with the data presented, which is a layer I soon will return to. The layer is presented in figure 33 and figure 33. However, if you have not checked the button, you should also provide either BAT coordinates, XRT coordinates, or both. If you forget, a popup window will inform you what you might need to do before continuing. The coordinates and the error should be given in degrees. Clicking submit will start the full pipeline in the background. The pipeline that runs, is the whole PyNOT pipeline, reducing images and comparing with the Gaia, 2MASS, and WISE catalogs. Besides, it also runs the point source selection pipeline and provides information of point sources and does cross-matching between the catalogs. After the pipeline is done the first layer closes and a layer with the most basic information appears.



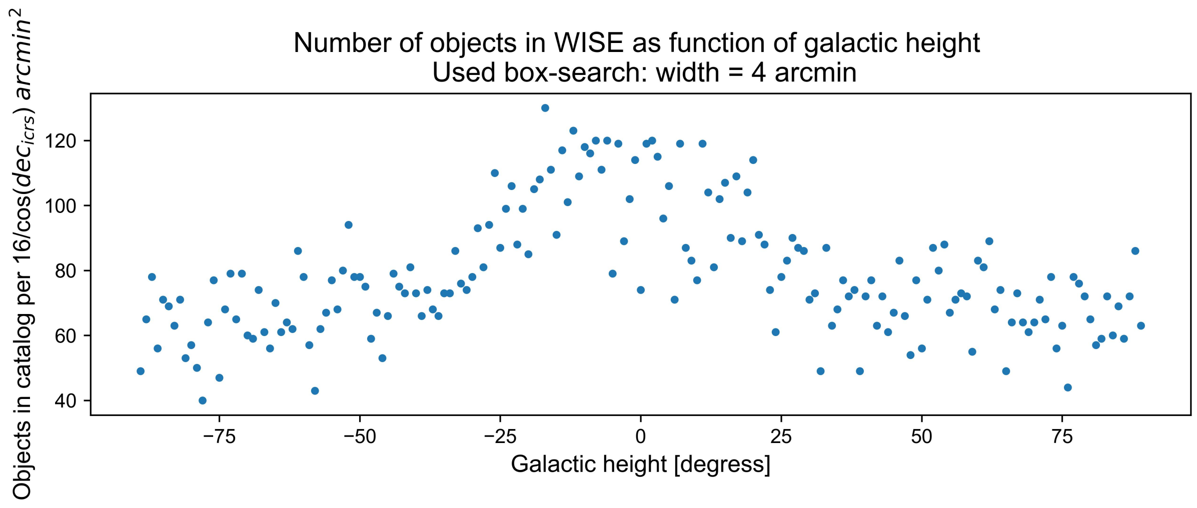


Figure 25: *The figure at the top shows the probability there will be an unwanted match between a transient placed at random and a source in the WISE catalog as a function of galactic height. It is considered a match if the randomly placed source is within a radius of 1.5 arcseconds of a source in the catalog. The second figure shows the respective number of sources in the catalog. The figures show there is a clear correlation between the density of sources and an unwanted match. Each data point in the top figure is the mean from simulating 20.000 random transients 50 times. The errors are the standard deviation of the mean. A slight bulge is seen in the catalog towards zero degrees. Besides this bulge, the number of objects is quite steady across all angles. It is also worth noticing that the y-axis in the second figure is a function of DEC in ICRS, found if transforming between the coordinate systems, due to how the catalog is queried.*

4.4.2 Timing window

While the pipeline runs, a window shows. The window was supposed to show how much time has passed and the time spend on each part of the pipeline such that the user could follow the process. However, it freezes, and it seems like the program has crashed even though it has not. It was found to be due to Python only handling one process at a time sequentially, leaving the GUI unresponsive while the pipeline is running. It will look like it has crashed until the pipeline has finished. I tried fixing it with multi-threading. However, it had a huge impact on the run time of the data processing as

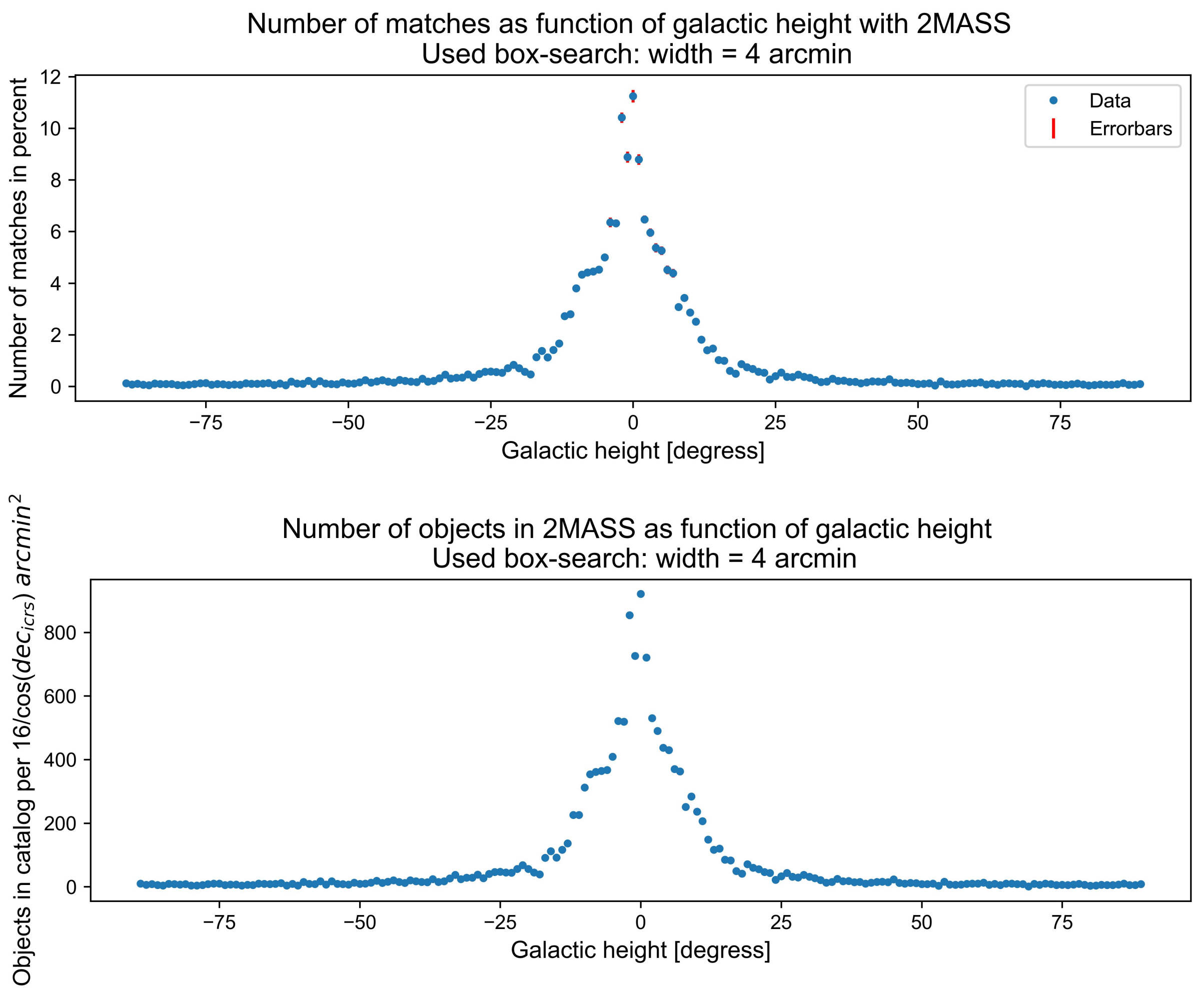


Figure 26: *The figure at the top shows the probability there will be an unwanted match between a transient placed at random and a source in the 2MASS catalog as a function of galactic height. It is considered a match if the randomly placed source is within a radius of 1.5 arcseconds of a source in the catalog. The second figure shows the respective number of sources in the catalog. The figures show there is a clear correlation between the density of sources and an unwanted match. Each data point in the top figure is the mean from simulating 20.000 random transients 50 times. The errors are the standard deviation of the mean. The 2MASS catalog has a sharp peak when looking at zero degrees GH. The number of objects is declining fast as the angle is changed. At more than 25 degrees from zero, almost no change in the number of sources is found. It is also worth noticing that the y-axis in the second figure is a function of DEC in ICRS, found if transforming between the coordinate systems, due to how the catalog is queried.*

both the timer class and pipeline use the same core. For that reason, multiprocessing was also tried, but not successfully, and this layer of the GUI was left as it was. The layer can be seen in figure 31. In this figure, the layer can be seen while the pipeline is running(left), and how it was supposed to look when it was done (right).

4.4.3 Basic information layer

After the pipeline is done you will be presented with a layer I call the $"Basicinformationlayer"$. This layer is a small pop-up window containing the most basic and important information found when compared with the Gaia catalog in the r-filter. This window is smaller as it should more be looked at

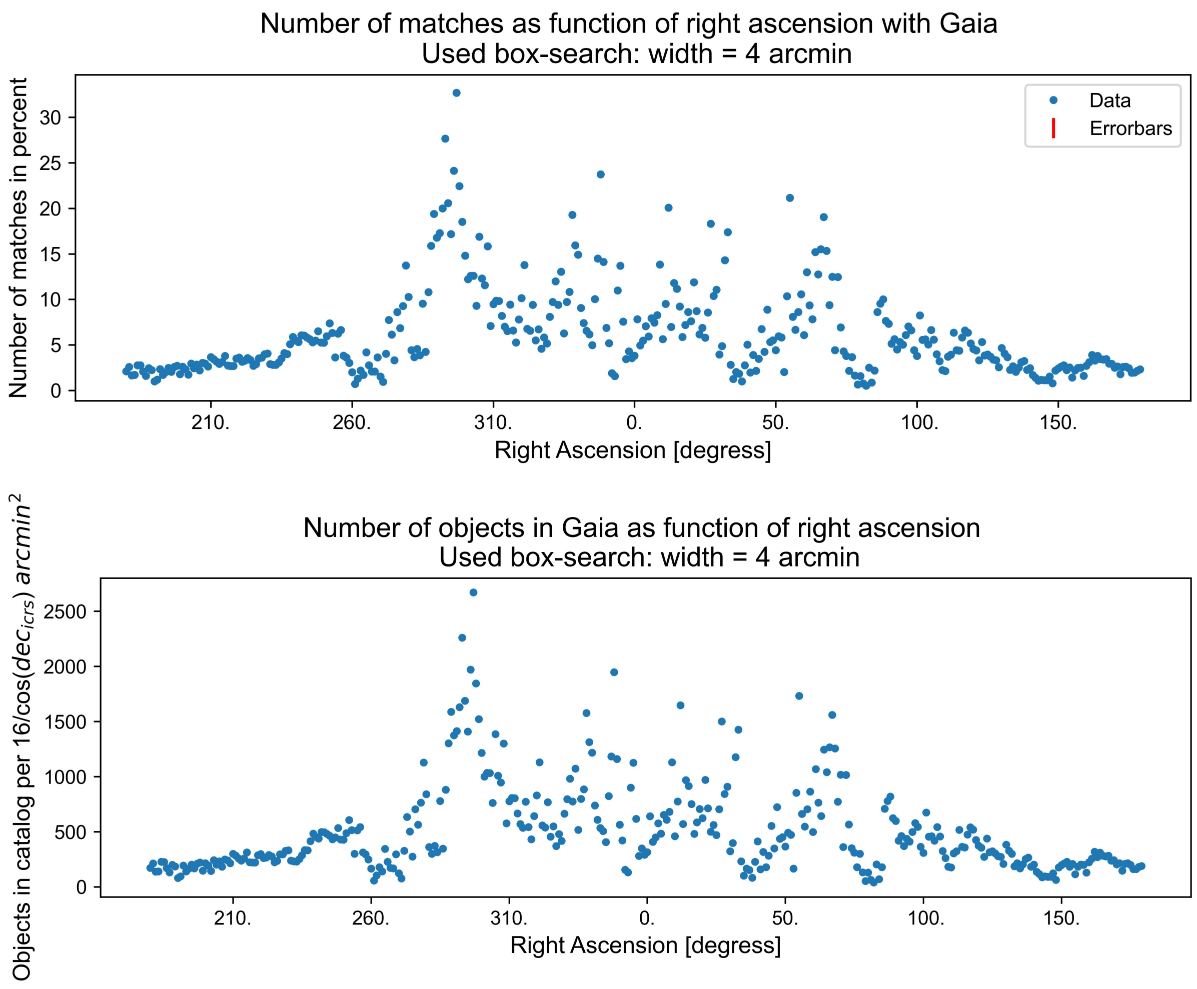
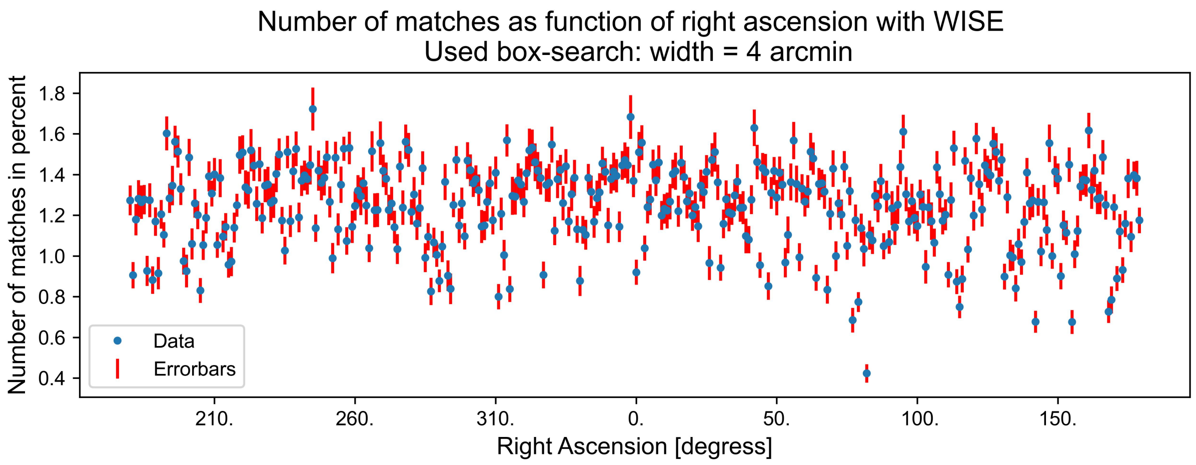


Figure 27: *The figure in the top shows the probability there will be an unwanted match between a transient placed at random and a source in the Gaia catalog as a function of right ascension. It is considered a match if the randomly placed source is within a radius of 1.5 arcseconds of a source in the catalog. The second figure shows the respective number of sources in the catalog. The figures show there is a clear correlation between the density of sources and an unwanted match. Each data point in the top figure is the mean from simulating 20.000 random transients 50 times. The errors are the standard deviation of the mean. Looking toward zero degrees, the number of objects found varies greatly. This might be due to dust in the plane. It is also worth noticing that the y-axis in the second figure is a function of DEC in ICRS, found if transforming between the coordinate systems, due to how the catalog is queried.*

as something for a very quick overview. The window is made color coded such that red is against it being the transient while green is for it being the transient. This way it should be fast for the eye and easier for the observer to quickly make a judgment upon which objects should have the most attention, without disregarding any.
If the observer wants to inspect the images further they can either look into the folders which have been sorted, or they can click the "*More info*" button, which will open the "*Info layer*".

4.4.4 *Info layer*

The last window, called the "*Infolayer*" shows more detailed information. In this window, you can also scroll through the images created with each catalog, as well as you can choose to only see point



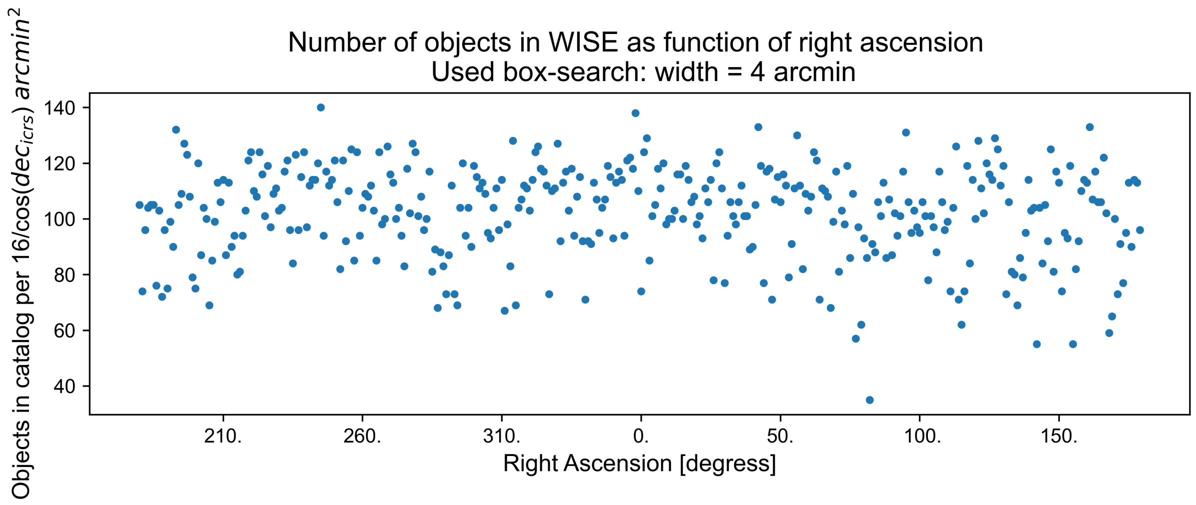


Figure 28: *The figure in the top shows the probability there will be an unwanted match between a transient placed at random and a source in the WISE catalog as a function of right ascension. It is considered a match if the randomly placed source is within a radius of 1.5 arcseconds of a source in the catalog. The second figure shows the respective number of sources in the catalog. The figures show there is a clear correlation between the density of sources and an unwanted match. Each data point in the top figure is the mean from simulating 20.000 random transients 50 times. The errors are the standard deviation of the mean. It appears there is no correlation between the number of sources in the WISE catalog and the RA. It is also worth noticing that the y-axis in the second figure is a function of DEC in ICRS, found if transforming between the coordinate systems, due to how the catalog is queried.*

sources in the plot. If there are other filters available you can also choose which filter you want to inspect. Besides the images, their respective information tables are presented aside. This should make it very easy for an observer to inspect the information gathered from the images.

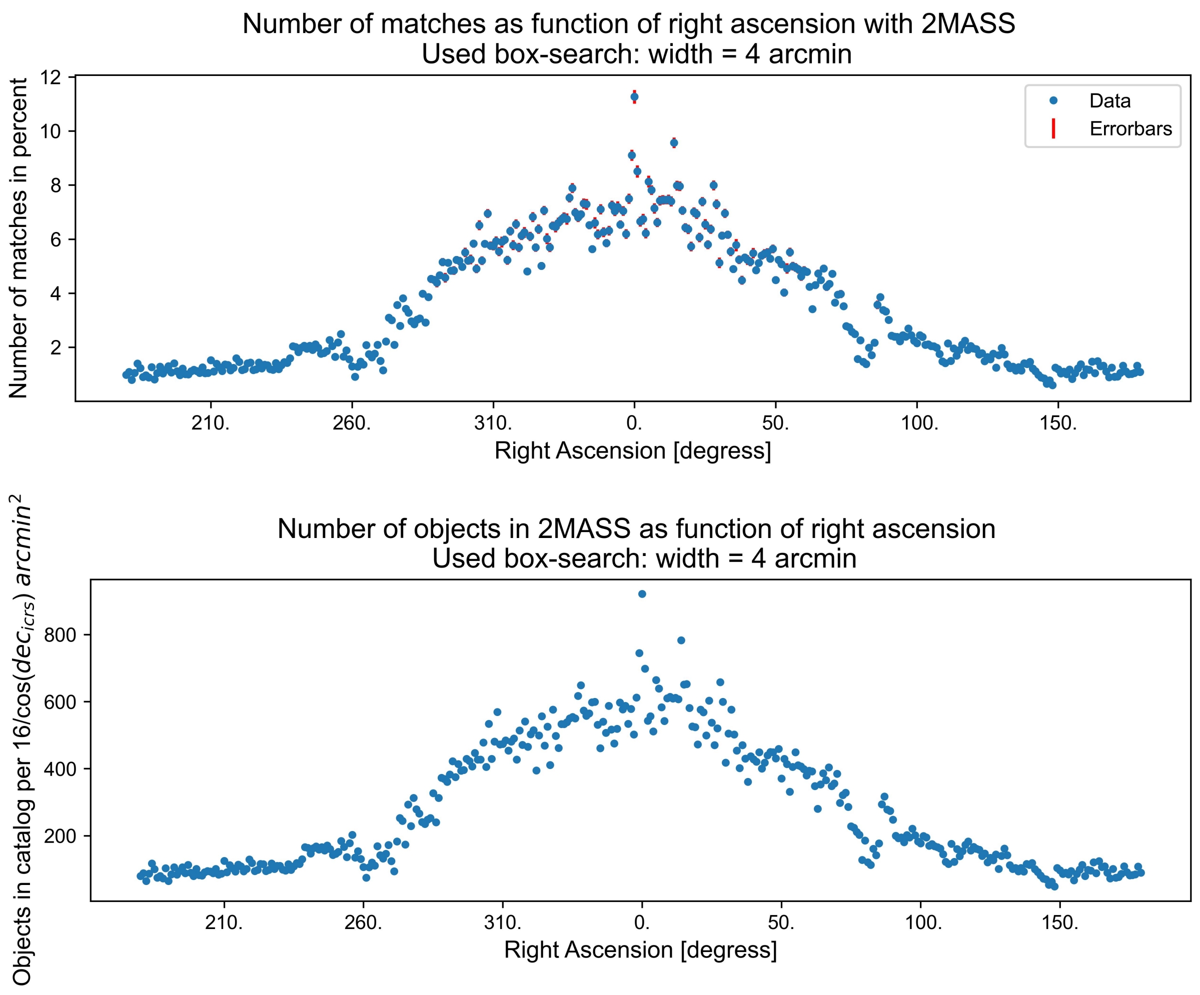
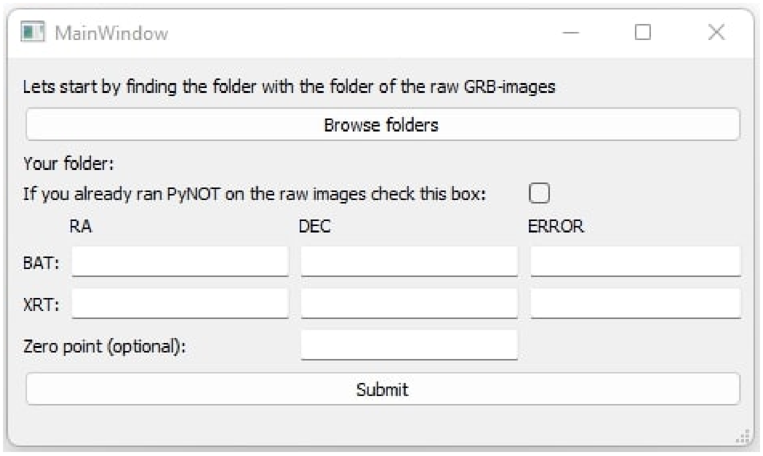


Figure 29: *The figure in the top shows the probability there will be an unwanted match between a transient placed at random and a source in the 2MASS catalog as a function of right ascension. It is considered a match if the randomly placed source is within a radius of 1.5 arcseconds of a source in the catalog. The second figure shows the respective number of sources in the catalog. The figures show there is a clear correlation between the density of sources and an unwanted match. Each data point in the top figure is the mean from simulating 20.000 random transients 50 times. The errors are the standard deviation of the mean. A bulge is found toward the center from 310 to 100 degrees. It is also worth noticing that the y-axis in the second figure is a function of DEC in ICRS, found if transforming between the coordinate systems, due to how the catalog is queried.*



MainWindow

Lets start by finding the folder with the folder of the raw GRB-images

Browse folders

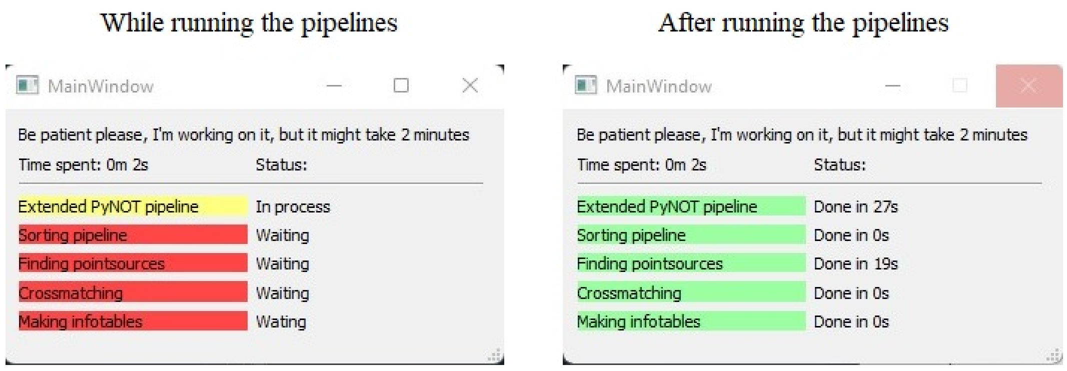
Your folder:

If you already ran PyNOT on the raw images check this box: ☐

	RA	DEC	ERROR
BAT:			
XRT:			
Zero point (optional):			

Submit

Figure 30: *Here the front layer of the GUI is shown. This layer takes the basic information as input, used for the PyNOT pipeline. RA, DEC, and ERROR for BAT and XRT coordinates should be given degrees. At this layer, you can also find your folder with your data.*



While running the pipelines

MainWindow

Be patient please, I'm working on it, but it might take 2 minutes

Time spent: 0m 2s	Status:
Extended PyNOT pipeline	In process
Sorting pipeline	Waiting
Finding pointsources	Waiting
Crossmatching	Waiting
Making infotables	Wating

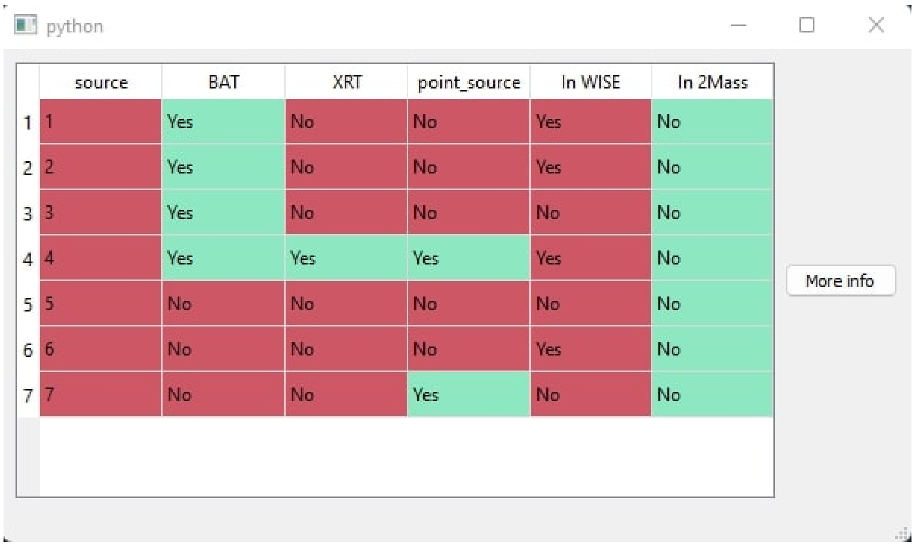
After running the pipelines

MainWindow

Be patient please, I'm working on it, but it might take 2 minutes

Time spent: 0m 2s	Status:
Extended PyNOT pipeline	Done in 27s
Sorting pipeline	Done in 0s
Finding pointsources	Done in 19s
Crossmatching	Done in 0s
Making infotables	Done in 0s

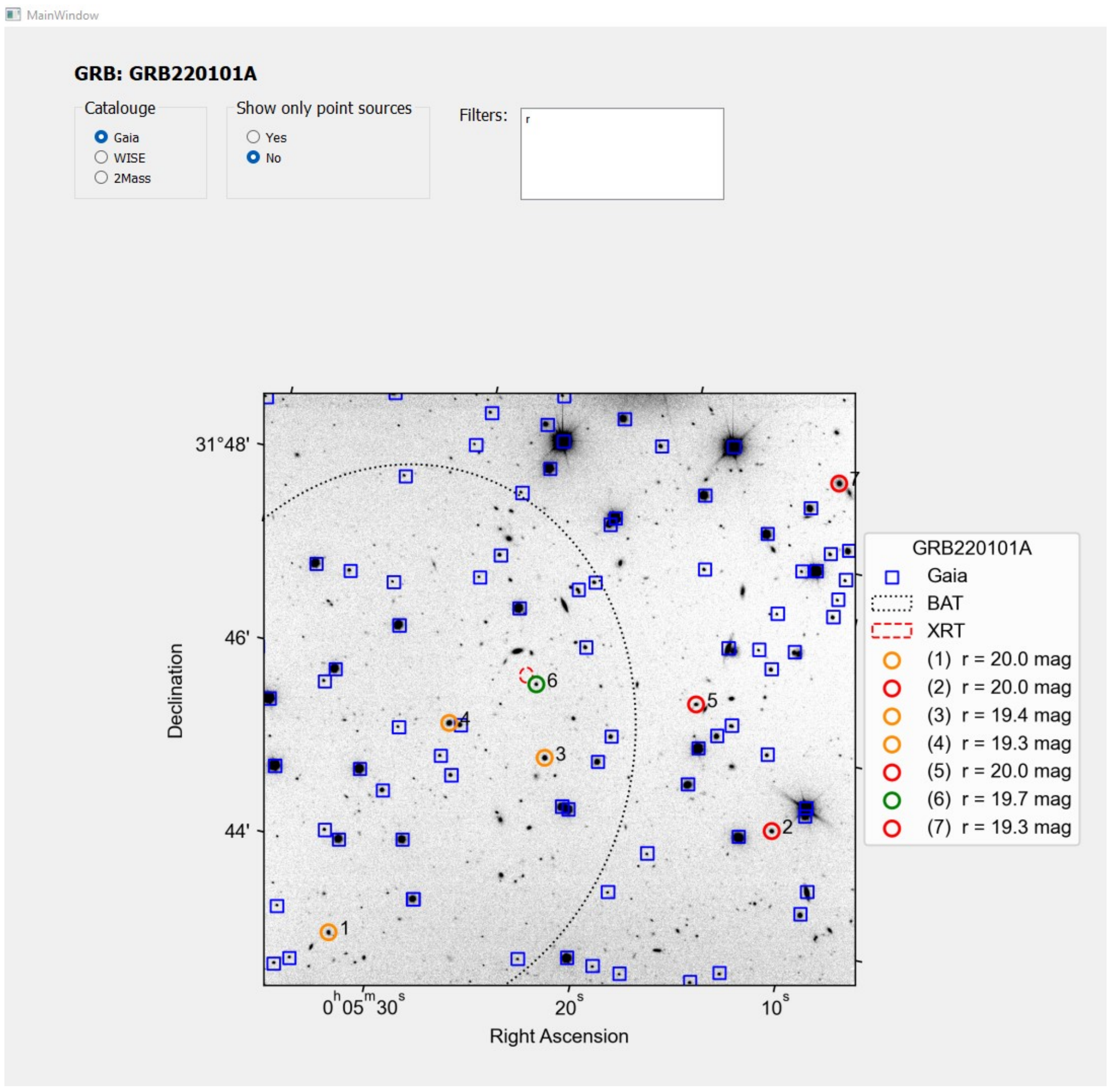
Figure 31: *This figure is the timing window. It did not work properly. It should, when the pipelines are run, show the image to the left and update it while running. After the data processing is done, it should result in the image to the right. The "Time spent" label is not updated either. However, this window freezes and it will become unresponsive until data processing is done.*



python

	source	BAT	XRT	point_source	In WISE	In 2Mass
1	1	Yes	No	No	Yes	No
2	2	Yes	No	No	Yes	No
3	3	Yes	No	No	No	No
4	4	Yes	Yes	Yes	Yes	No
5	5	No	No	No	No	No
6	6	No	No	No	Yes	No
7	7	No	No	Yes	No	No

Figure 32: *This is the small "Info layer" of the GUI. It is providing the most basic information from the r-filter. In this example, the data is from GRB220101A. The table provides information about whether a source is consistent with BAT, XRT, if it is a point source and if it is found in the 2MASS or WISE catalog. It is color coded such it should be easy to review fast.*


MainWindow

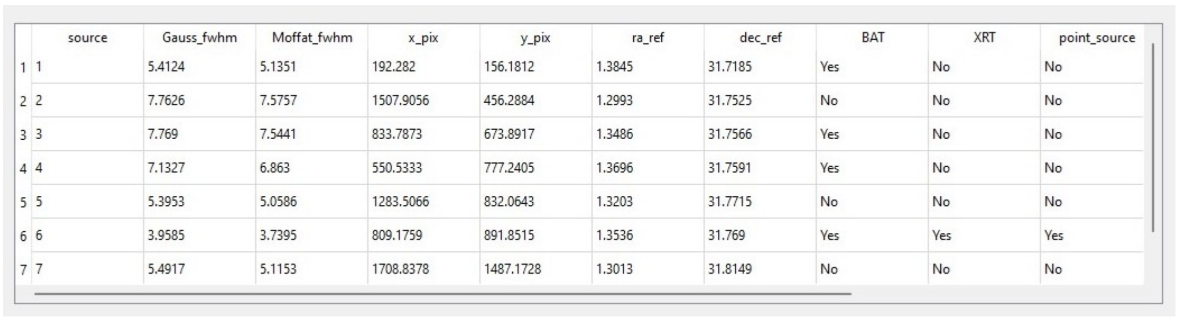
GRB: GRB220101A

Catalouge: [✓ Gaia] WISE 2Mass

Show only point sources: Yes [✓ No]

Filters: r

Figure 33: *This is the left side of the "Information layer" of the GUI. The right side is shown in figure 34. The "Information layer" is providing information from the transient detection like the "Info layer", but it also includes the coordinates in the WCS. In this layer, it is also possible to choose which catalog and filter you want to be displayed as well as the corresponding data table. Further, you can choose to only see the point sources displayed. In this figure of the left side, the image from transient detection is displayed. Here the options are also shown.*



	source	Gauss_fwhm	Moffat_fwhm	x_pix	y_pix	ra_ref	dec_ref	BAT	XRT	point_source
1	1	5.4124	5.1351	192.282	156.1812	1.3845	31.7185	Yes	No	No
2	2	7.7626	7.5757	1507.9056	456.2884	1.2993	31.7525	No	No	No
3	3	7.769	7.5441	833.7873	673.8917	1.3486	31.7566	Yes	No	No
4	4	7.1327	6.863	550.5333	777.2405	1.3696	31.7591	Yes	No	No
5	5	5.3953	5.0586	1283.5066	832.0643	1.3203	31.7715	No	No	No
6	6	3.9585	3.7395	809.1759	891.8515	1.3536	31.769	Yes	Yes	Yes
7	7	5.4917	5.1153	1708.8378	1487.1728	1.3013	31.8149	No	No	No

Figure 34: *This is a part of the right side of the "Information layer" of the GUI. The left side is shown in figure 33. In this Layer, it is also possible to choose which catalog and filter you want to be displayed as well as the corresponding data table. Further, you can choose to only see the point sources displayed. On the right side of the layer, the data table is presented, which is seen here.*

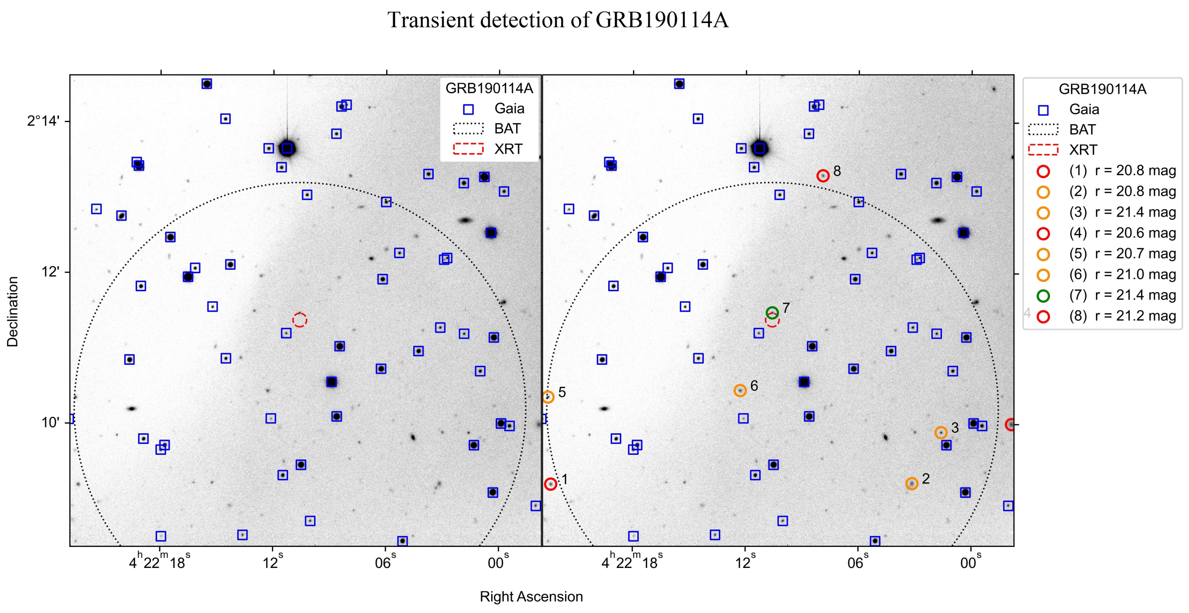


Figure 35: *Results from transient detection of GRB1901014A. In the figure to the left, the default magnitude threshold of 20.1 is used while the magnitude threshold is raised to 21.5 in the right figure. With the higher threshold, a transient is found to be consistent with both BAT and XRT and is marked green in the figure to the right. In both images, the background seems to be distorted. This distortion is believed to be caused by the BIAS files. One of the BIAS files can be seen in Appendix 4.*

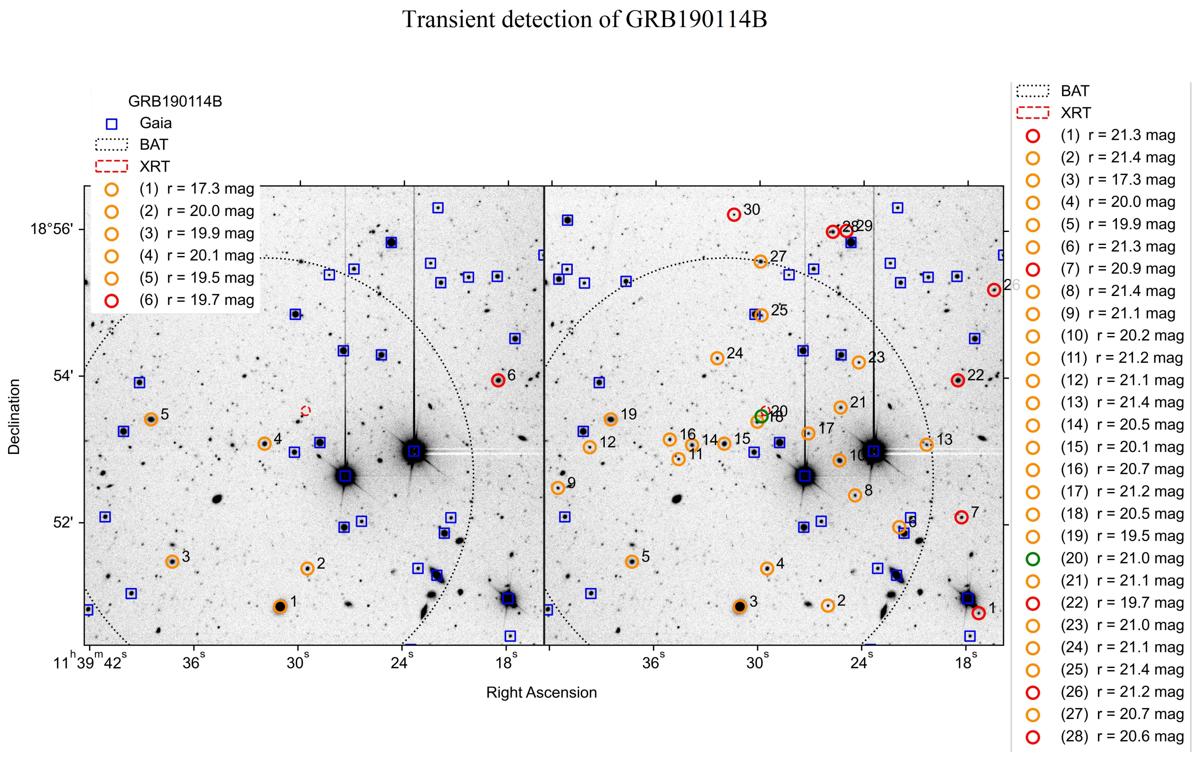


Figure 36: *Results from transient detection of GRB1901014B. In the figure to the left, the default magnitude threshold of 20.1 is used while the magnitude threshold is raised to 21.5 in the right figure. With the higher threshold, a transient is found to be consistent with both BAT and XRT and is marked green in the figure to the right. Raising the magnitude threshold results in too many sources for PyNOTs image having the full legend saved.*

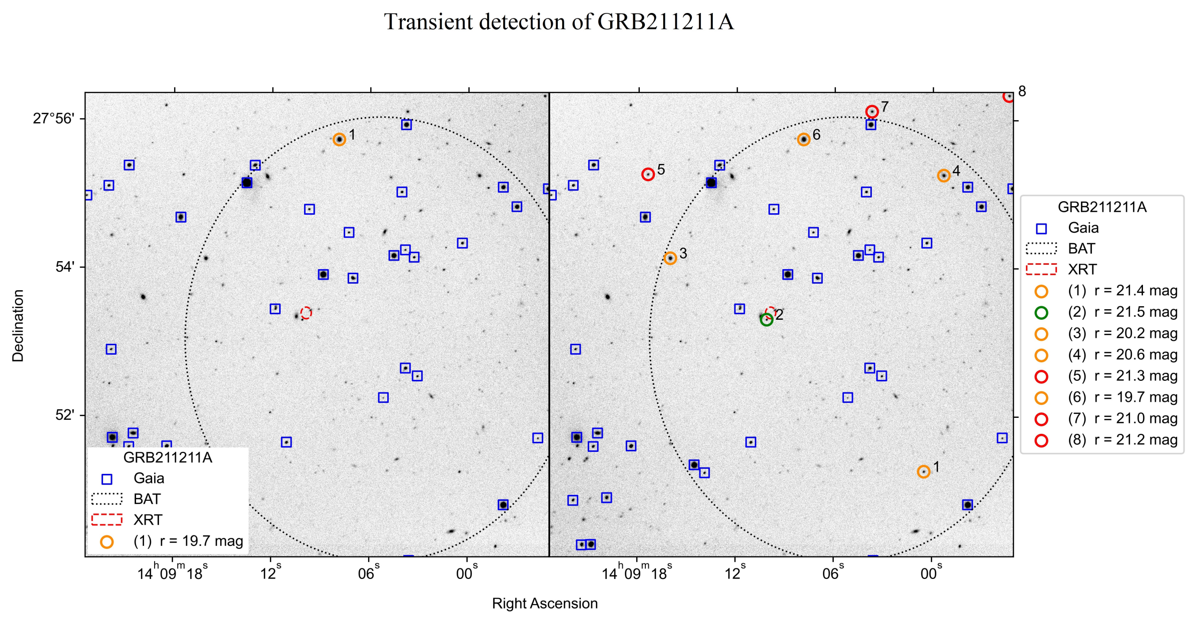


Figure 37: *Results from transient detection of GRB211211A. In the figure to the left, the default magnitude threshold of 20.1 is used while the magnitude threshold is raised to 21.5 in the right figure. With the higher threshold, a transient is found to be consistent with both BAT and XRT and is marked green in the figure to the right.*

5

DISCUSSION

5.1 Initial tests of PyNOT

5.1.1 Installation and usability of PyNOT

Initially, I was not able to install PyNOT. However, Jens Krogager helped resolve the issues by updating the source code of PyNOT. PyNOT did not rely on any other programs being installed except for Python, which made installation easy after the initial problems were fixed.

Moreover, PyNOT was quite easy to use as the documentation, in the form of a README file, was okay to use. However, it could use a bit more information. A concrete example sample could be useful for a person using PyNOT for the first time. A problem I found, was that PyNOT has the functionality to find and calibrate zero point itself. Therefore, it is optional whether you will provide a zero point for PyNOT to use. If you decide to let PyNOT figure it out on its own, a problem occur if PyNOT does not manage to calibrate the zero point. For that reason, you will have to run PyNOT again. This time providing a zero point yourself. But, doing so will resolve in another error. To fix this error, you will have to delete the files PyNOT has already made in its first run, before running the pipeline again. You could ignore the *"Initialize"* function, which is the first step in the PyNOT pipeline. However, this part took less than a second for all the data sets I tested. So, it is easier to delete all the files and folders, except the folder containing the raw files. This is not optimal, as fast transient detection requires to be fast and this is wasting precious time. This must be done every time you would like to run the PyNOT pipeline again on the same data. Meaning, every time I had made changes to PyNOTs source code or changed some parameters to get different results, I had to go through this process of deleting stuff manually. This is something that would make the experience much better if fixed. It would both make PyNOT easier to use and provide a better workflow for the user. The zero point PyNOT uses is a bit unclear if it is in counts per second or in counts in the images. I suspect it is counts in the image, not adjusted for exposure time, as it was found that the number 27.5 is a good number, whereas a zero point of 25 counts per second is what is found by NOT [37]. Therefore, 27.5 counts in an image with an exposure time of a typically 300 seconds would be a good guess for a zero point when using images from NOT. This will result in a slight offset in the estimated magnitudes of the objects. But this trade-off, is worth it, as in the matter of fast transient detection, the location is more important than accurate magnitude estimation. Accurate magnitude estimation can be done after the exact position is determined. Additionally, for fast transient detection, it is not reliable to use PyNOTs automatic zero point calibration. In two out of the five data sets, PyNOT did not estimate a zero point. This resulted in the pipeline having to be run again. For that reason, the function is not robust enough. I did use it in the cases where PyNOT was able to find the zero point. However, in a situation where time is an important factor, I would not rely on that function as it is now.

It was found that PyNOT could run most of the data sets provided by NOT. As seen in table 1 in the result section, I was not able to run PyNOT with the data set from GRB190829A. PyNOT gave the error "Value Error", which is an error PyNOT is not able to handle. As seen in table 1, I was neither able to run PyNOT with the data set from GRB191019A. It should be noted that an error message was provided here as well. The error message explained that there must be at least three BIAS files in the data set to be combined. These files are needed for data reduction, and if knowing in advance that PyNOT requires this, the problem is easily solved. It was not written in the documentation, but if it was, it would be fair to not count this one as a data set PyNOT could not use. This is because PyNOT, like any other program, needs the correct input. For that reason, it should be added to the documentation. It would be a shame if imaging were done only to discover afterward that PyNOT can not be used because requirements were not fulfilled. To sum it up, PyNOT was able to use five of the seven data sets from NOT. Of those five events PyNOT did complete, it took an average of 33.13 seconds for the full pipeline to complete, which is acceptable.

5.1.2 PyNOTs magnitude threshold

When the default magnitude threshold at 20.1 was used, PyNOT managed to identify the transient in GRB190114C and GRB220101A. However, it was not able to identify the transient in GRB190114B, GRB211211A, and GRB190114A with the same threshold. This magnitude threshold was found to be too low if you want to be sure the transient is found in the image. When the threshold was raised to 21.5, the transients were detected in all five events in the r-filter. This indicates, that PyNOT is more useful for transient detection if a higher magnitude threshold is applied. Thus, the default threshold can not be recommended to use for fast transient detection as it is not robust enough. Consequently, more candidates for being transient are left in the final image, as those fainter objects are not present in the Gaia catalog. Moreover, finding the right threshold to use will vary from event to event. Hence, a too high magnitude threshold can be chosen, resulting in an excessive number of candidates being present in the image. It was found that a small change in the magnitude threshold can make a difference in the number of candidates and this threshold is thus quite sensitive. Inspecting the images of GRB190114C in the different filters shown in figure 13, reveals the transient is the brightest of the candidates in all the filters. This indicates an anticipated knowledge of the event would, at least in this case, have led to only one transient candidate being present. This candidate would also be the actual transient. Looking at the image of GRB220101A in figure 5, it shows the slight change in magnitude threshold reduces the number of candidates. However, the transient source is not the only candidate left in the image. Lowering the threshold to that of the transient in the image of GRB220101A would have kept six of the eight possible candidates. Nevertheless, for an event, the foreknown knowledge of the magnitude of the transient is difficult to guess. For that reason, it is still better to use a too high magnitude threshold. It will leave more distracting sources to decide between. However, it is better than the actual transient not being detected at all.

The magnitude threshold created a bit of confusion at first. It would be better if PyNOT marked sources with the magnitude above this threshold instead of removing any indication the source has been detected. At the moment it appears in the final image that a source with a magnitude above this threshold was not detected in the reduced image, which is not the case. I thought PyNOT would simply not compare the sources with the Gaia catalog. For this reason, I believed for a very long time that the transients were not actually present when object detection was done, even though they were. However, the magnitude threshold removes all indications of objects being detected if fainter than this threshold. This information about the individual parameters would be good to have in the documentation as well.

5.1.3 Reviewing PyNOTs current methods for transient detection

A strong feature of PyNOT, is that it supplies error circles from the BAT and XRT coordinates. PyNOT uses color coding to tell if a source is within these error circles. It gives an indication of whether a source is consistent with the coordinates. If a transient is marked green, it is within five sigma of the center of the XRT coordinate. This makes many of the sources outside very unlikely to be the transient and it is a very strong indicator from PyNOT. But in some cases, two sources can both lie within the XRT circle and you would then not be able to identify which of them are the transient. It might also be that XRT coordinates are not accessible at first. This could be due to SWIFT taking up to about 100 seconds to swivel for the XRT. So, for fast transient detection, we might have to rely on the BAT error circle. Those sources consistent with BAT are marked yellow. However, as the error circle is much larger, the feature loses its strength. The feature is not useless, but often more sources will be consistent with BAT at the same time. Thus, it would be useful to get rid of, or at least get extra information about the transient candidates to narrow down the actual transient.

PyNOTs method for transient detection, using coordinates of objects in images and comparing those with coordinates found in catalogs, was tested. Figure 24 shows, if trying to use the method of comparing detected sources with catalogs on a transient laying in the line of sight with the center of our galaxy, it is not very efficient. For a random positioned transient at a galactic height of -4 degrees, the probability of having a match in the Gaia catalog is more than 70 percent. However, figure 24 show, if looking at an object about 20 degrees down or up from the galactic plane, the probability of a match is about 10 percent. Thus, an unwanted match is much less. Further, figure 24 also shows that the probability of a match drops quite fast to around one percent if looking at more than plus minus 25 degrees. Zooming in on the y-axis of the second image in 24, it is revealing that an image taken at more than 50 degrees away from the plane will have less than 40 sources per 4 $arcmin^2$. This is clearer in figure 38, where the y-axis has been zoomed in.

Moreover, the test revealed that something is happening at zero to 10 degrees in galactic height in the Gaia catalog (Figure 24). This effect is not seen in the 2MASS or the WISE catalog (Figure 25 and 26). A good explanation could be that the drops are caused by dust in the plane of the galaxy. The 2MASS and the WISE catalogs look further into the infrared where extinction is less prominent. Hence, The Gaia catalog would be the catalog that is expected to be impacted the most.

Looking at figure 26 that shows the test of 2MASS, we see a similar pattern as in Gaia. However, the catalog has fewer sources at all angles except at exactly zero degrees. It should be remembered the catalogs are covering different wavelengths. Hence, they are not directly comparable in the number of objects. Looking at figure 25, it shows that the WISE catalog has fewer sources toward the plane than 2MASS. Further out, WISE has more sources than both 2MASS and Gaia. This indicates that in the near-infrared where Gaia has its shortcomings, a mix of WISE and 2MASS might be better than only using one of them. As the wavelengths they cover are only slightly different, they would probably

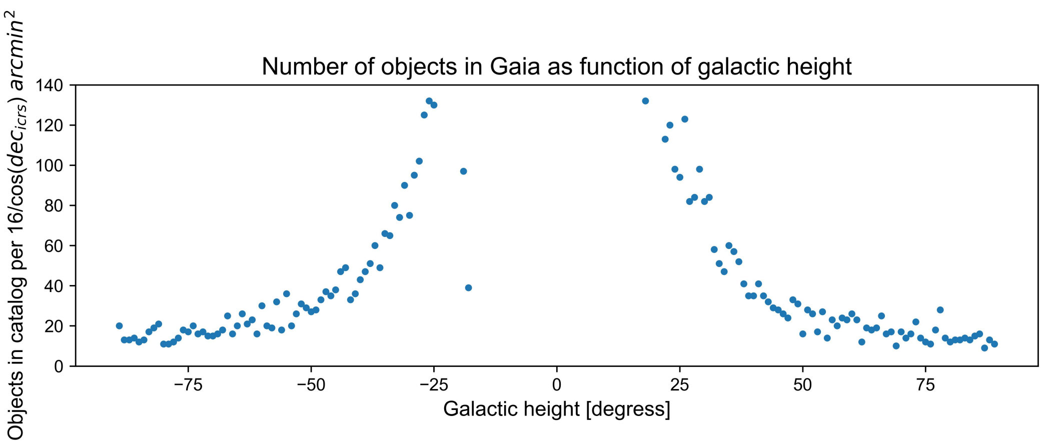


Figure 38: *This figure is a zoom in of 24. In this figure, it is more clear the number of sources found in the Gaia catalog at higher degree angles.*

provide better coverage of the whole sky if used together. Moreover, they would also cover a broader range of wavelengths.

The same test was carried out for right ascension. For Gaia, the results seemed to vary. If observing at less than 100 degrees from zero, the probability of a match varied between five and 30 percent. Thus, images taken in this area would show fluctuating results. This is most likely mainly due to extinction caused by dust in the galactic plane. The results would be more stable if the images are taken at more than 100 degrees from the center. At these degrees, the probability of a match is less than 10 percent. For WISE, there seems to be no correlation with the change in right ascension. However, 2MASS shows a bugle towards the center peaking at a 12 percent probability of a match. The bulge softly declines until 100 degrees away from the center. At more than 100 degrees away from zero, the probability of a match is almost flat with a probability of a match less than three percent.

All the tests show, that there is a direct correlation between the number of sources in the catalog and the probability of a match. This logically makes sense as the density of objects is higher if more sources are present. It suggests, that in regions that are very dense in objects, the use of coordinates might not be the strongest method. However, the method is not necessarily of no use. I tested fading objects in the images of GRB220521A. However, as GRB220521A is laying in the direction of the galactic plane, I also did a short run of the PyNOT pipeline with this data set. It shows that even though the Gaia catalog has many sources and there were more than 100 candidates, the transient is found (Figure 39). This proves the strength of the method. Nonetheless, this image is still not taken directly toward the center. Meaning a worse case is expected that might be even more difficult to decipher and extract information from.

Some strange things are worth pointing out that were discovered when making the image shown in figure 39. In all the tests I have done with PyNOT, I have kept the angular distance threshold at the default 1.5 arc seconds. This seemed to work fine for the five events I have tested with PyNOT. However, when I ran the pipeline on GRB220521A, the resulting image is the one seen in figure 40. This is a huge mess, and getting information out of that is quite difficult as some might imagine. It may also give the false belief, that there are more sources not in the catalog than what is true. To go from the image in figure 40 to the image in figure 39, I changed the threshold of angular distance to be

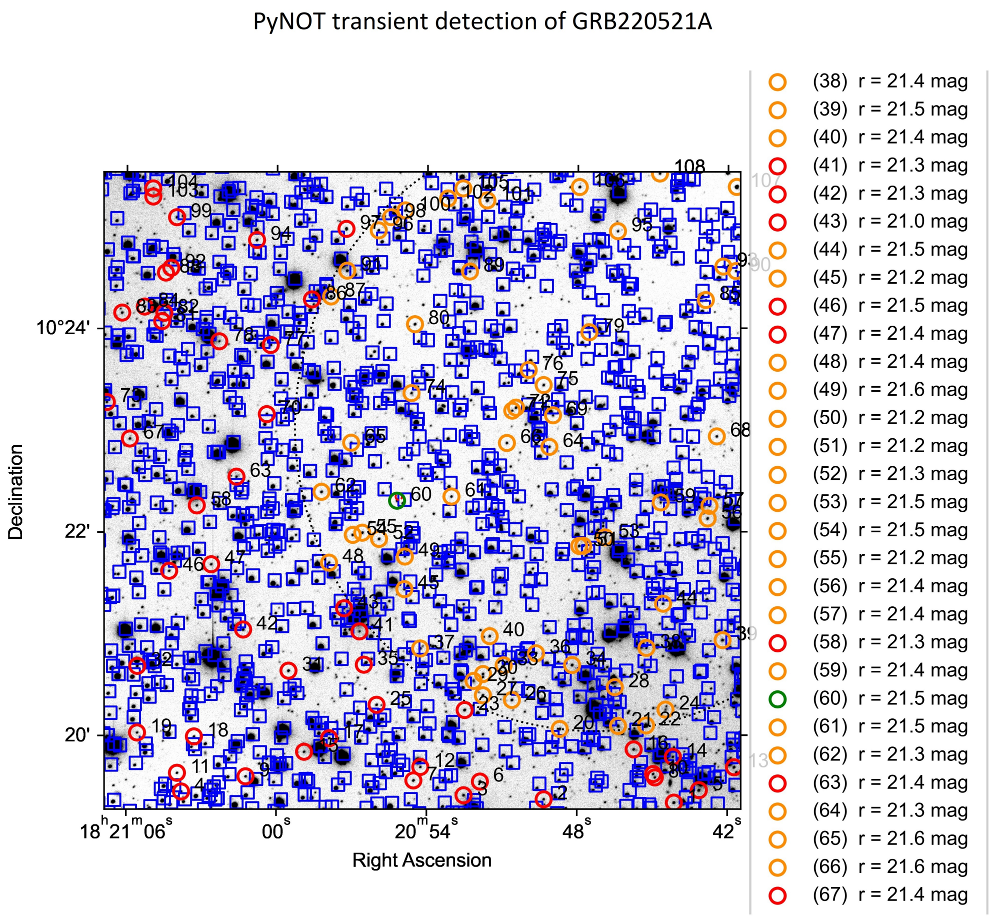


Figure 39: *This is the image PyNOT creates of GRB220521A with an angular distance threshold of four arcseconds and a magnitude threshold of 21.6. This figure shows that many sources are in the area of GRB220521A. To create this image, only the first three of the four OBJECT images from the data set are used. It should be noted that the legend is too large to be kept in this image due to the high amount of transient candidates. The transient is found to be consistent with both BAT and XRT. The transient is marked with green and as source number 60.*

four arcseconds. I did not figure why this threshold had to be changed for this particular GRB event. It might be something worth looking further into. This data set is also not part of the PyNOT test, as I first got it later. It was intended for testing if detecting fading transients would be possible. Moreover, for figure 39 and 40, I had made changes to which raw files were left in the folder I originally was provided. Because of that, it is not included in the PyNOT test. However, I would not ignore the discovery of the threshold for the angular distance.

Another thing was discovered when working with GRB220521A. The raw data in the r-filter consisted of four OBJECT files. In the latest of them, the transient had faded too much to be detected in the image. This resulted in PyNOT not being able to find it even though the magnitude threshold was raised to 25. It was found in the three other images. I had to manually remove the latest image for the transient to be found. Meaning, I could not find the transient if not doing further investigation myself, defying the purpose of an automatic tool. Hence, a feature for finding objects missing in later images and not reappearing could be implemented into PyNOT.

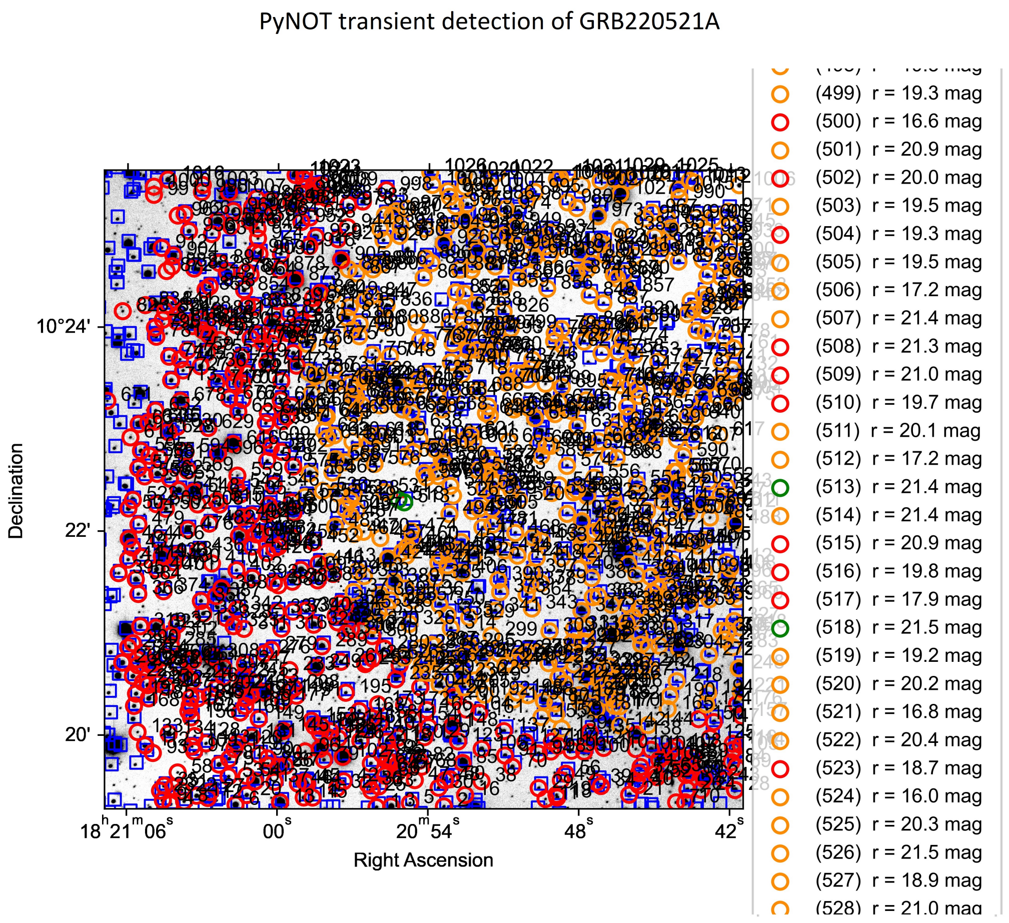


Figure 40: *This is the image PyNOT creates of GRB220521A with a magnitude threshold of 21.6 and PyNOTs default angular distance threshold of 1.5 arcseconds. In this image, it shows that very few sources are matched with the sources in the Gaia catalog. To create this image, only the first three of the four OBJECT images from the data set are used. In this image, two sources are marked with a green circle as they are found to be consistent with BAT and XRT. The sources marked with an orange circle are those sources that are consistent with BAT. Those sources that are neither consistent with BAT nor XRT are marked with a red circle. The blue squared are the sources in the Gaia catalog.*

A third finding was made for GRB220521A. PyNOT created two folders in the imaging folder for GRB220521A. I believe this is because the imaging of the event was taken in the late evening and the imaging crossed midnight. Hence, it was taken across two different dates. If not knowing, this is quite confusing. Further, images in the z-filter were taking at different dates which resulted in those files being separated. Instead, those files should have been combined. Because of this, I did not take that into account when creating the point source pipeline as I was not aware of the problem.

5.2 Expanding PyNOT to have more functions

5.2.1 The use of more catalogs

Narrowing down the transient among several transient candidates leads to one of two uses for expanding PyNOT to use other catalogs than the Gaia catalog. Expanding PyNOT such that it also uses the 2Mass and WISE catalog, showed that using 2Mass is inutile, at least in optical images as it did not really contribute to a more complete overview. This can be seen by investigating the images in the appendix. The WISE catalog on the other hand, did in some cases help provide a more complete sample of known sources. An example of this can be seen in figure 41, where the Gaia and WISE catalog has been used for GRB190114A. Here, the sources labeled number 1, 4, 5, 6, and 8 in the image made with Gaia to the left in figure 41, is to be found in the WISE catalog. This leaves only the sources labeled 2, 3, and 7 to be unknown. In the area of GRB190114A, the WISE catalog contained far more sources than the Gaia catalog. If holding this together with 24, 25 and 26, 27, 28, 29, it may appear some areas are better covered by WISE, especially at high galactic height and right ascension. However, many of the locations from WISE marked in figure 41 to the right, have no object in them. This is most likely due to the image being taken in the optical waveband, while those locations are from objects in the near-infrared. This indicates, that using WISE together with Gaia might be beneficial, but not a substitute. Even though it appears that more sources are in the WISE catalog in some areas, many of those sources would not be matched with objects found in the optical image. This is also clear looking at the image to the right in figure 41. This is also why fewer transients candidates are present in the image that used the Gaia catalog, even though the WISE catalog has more sources. However, the results indicate that the WISE catalog might be a good option for when the NTE is ready as it seems to have a good coverage at high galactic height and right ascension as seen in figure 25 and 28. Furthermore, the tests show it is possible to implement the use of any arbitrary catalogs into PyNOT. It also shows it would be a good upgrade for PyNOT such that it can be used for transient detection across a much broader spectrum making it even more versatile. In the near future, the data collected with the EUCLID[38] satellite may also be used with the NTE instrument in addition to 2MASS and WISE, as this also will make a sky survey in the near-infrared[38].

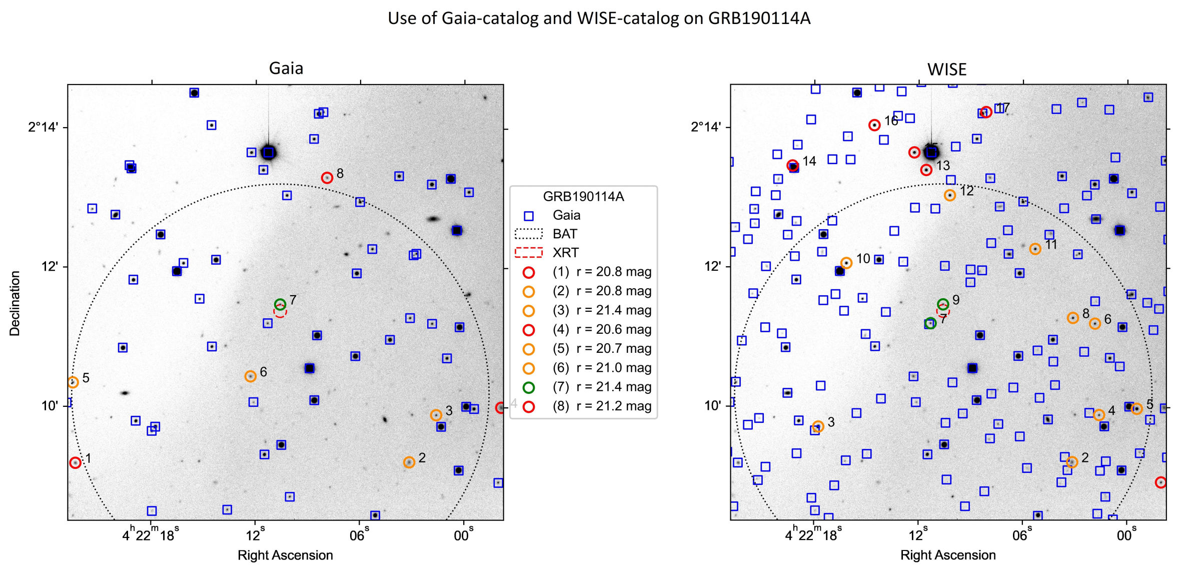


Figure 41: *This figure shows the images created with PyNOTs transient detection function if the Gaia catalog (left) and the WISE catalog (right) are used. It can be seen from the two figures that source number 1, 4, 5, 6, and 8 from the left image is accounted for in the right image. This indicates that the use of the WISE catalog if used in addition to the Gaia catalog, can contribute to a more complete overview, even in optical images.*

The changes made for using other catalogs were done inside of the source code of PyNOT. This could be modified such as downloading the other catalogs is done outside PyNOT while running the first two of the three functions in the PyNOT pipeline. However, if wanted to use other catalogs for transient detection in the infrared, it might be good two make a new file which has the same structure as PyNOTs ”transient.py”. Instead of using Gaia, it should instead use WISE or 2Mass for transient detection and for WCS calibration. Otherwise, it could be made more versatile, such as it could take an arbitrary catalog.

When testing the ”run_pipeline” function, where the catalogs are queried, there was very little impact on the run time. The run time before and after making the changes to PyNOT was on average 30 seconds within one second from each other. The catalogs take between five to 10 seconds each to download. Therefore, it would be expected the function would take at least 10 more seconds if the catalogs were queried in sequential. This proves the implemented threading to be highly effective as the impact is less than 0.5 seconds. However, the average total run time is increased by 93.6 percent. This was found to be a consequence of doing transient detection in three catalogs instead of one, impacting the *"find_new_sources"* function. Moreover, I have also changed the code in transient.py, such as it creates a PNG image, which also takes some extra processing time. Creating PNG-images seems to have a significant impact on the run time. Those data sets with more filters were especially impacted. To take from this, the transient detection would benefit from being executed with multiprocessing. Moreover, a lighter format than PNG, like JPG could be considered.

A note on the downloading of catalogs must be made. For most of the time, when querying the catalogs, they were downloaded fast. Typically, in the range of five to 10 seconds. Nonetheless, I did experience a period with very long query times. Table 8 is created with the *"find_new_sources"* function with the extension of using 2MASS and WISE as well. However, the catalogs are queried when doing transient detection and it is done in sequential order. Moreover, the extra PNG-image is not created either. Thus, we can see the individual times it takes to query and do transient detection for 2MASS and WISE. The run time for 2MASS and WISE are very high in this table. It was found that the time mostly was spent on fetching the catalogs. The data in table 8 is generated at around 22 o’clock on a Wednesday, which is important to notice as the times it took varied a significant amount depending on the time of the day. At some moments, the function did not manage to finish the job request to the server as it timed out. The problem seemed to resolve itself, and the long query times disappeared. However, table 8 suggests having the catalogs in a local server would save some time. Moreover, problems experienced with long query times for the catalogs would not occur.

5.2.2 The point source selection pipeline

To further narrow down the number of potential candidates, I managed to create a pipeline for point source selection. When used, it was found to be a robust method for narrowing down the number of candidates. In all the events, the actual transient was found to be a point source by the pipeline. In addition, the number of candidates was greatly reduced in the image with the Gaia catalog using the r-filter. In both GRB190114C and GRB220101A, the transient was the only candidate left in the image. In GRB190114A and GRB211211A only one other candidate besides the transient was kept. In GRB190114B there were initially found 30 candidates which was not in Gaia when using the r-filter. With the point source selection pipeline, this number was reduced to five candidates. Among

Event:	Gaia run time (pre-downloaded)	2MASS run time	WISE run time	Total run time
GRB190114A	1.61s	114.40s	55.10s	171.10s
GRB190114B	4.65s	219.62s	314.15s	538.41s
GRB190114C	2.96s	176.39s	213.95s	393.30s
GRB211211A	2.56s	207.06s	187.31s	396.92s
GRB220101A	0.92s	46.24s	46.34s	93.51s
Average times	2.54s	152.74s	163.37s	318.65s

Table 8: *This table is showing the run times of transient detection with the use of the Gaia, the WISE, and the 2MASS catalog. In those results, the Gaia catalog is locally on the computer used whereas the WISE and 2MASS catalogs have to be fetched from a remote server. What is worth noticing in this table is the very long runtimes when using the WISE and the 2MASS catalogs. It was experienced that there was a period where the runtimes varied a lot and it was found most of the time was spent on getting the catalogs. This problem disappeared by itself, but it indicates that having the catalogs locally is more robust.*

those five, was the transient one of them. For that reason, it seems like the method is quite effective for transient detection. Hence, a point source selection pipeline would be a strong addition if added into PyNOT. I managed to make the pipeline such that running it automatically was done in all the images created by PyNOTs transient detection function.

I optimized the pipeline by using Numpy functions whenever I found it meaningful. Further, I structured the code such that it would reuse the FWHM-list for each filter. I did so because calculating the FWHM for all the sources in the images is the dominating part when it comes to the total runtime of the pipeline. Meaning, for each filter, the FWHM list will be used for all the catalogs resulting in the pipeline being less impacted by using several catalogs. In the five GRB events, the pipeline took an average of 26.4 seconds to finish the point source selection of all the images from PyNOT. In the test, GRB190114B was the slowest by 54.3 seconds. The other events all took less than 30 seconds to finish. So besides for GRB190114B, the run time is also acceptable.

It was found that saving the final images had a huge impact on the run time. Not saving the images reduced the average run time to 18.13 seconds. The images from the point source selection were saved as PNG-files with a density per inch(dpi) of 300. This was chosen to be a good middle ground as it resulted in a resolution of 2700x1800 pixels. Moreover, if dpi was raised to 600 the average run time would be 45.28 seconds while GRB190114B would take one minute and 20 seconds to finish. This impact on run time, I did not find to be a tradeoff worth it. I did not, as I already found the images quite clear at 300dpi until you really zoomed in. And at this point, you would probably use SAOimageDS9 to further investigate the object. So, the image quality of 300dpi for locating the object is good enough. And locating the transient is the essential purpose of fast transient detection. In addition, a lower resolution would make the image grainy, and little speedup is gained by doing so. However, I did try, quite late in this project, saving the files as JPG-files, as I realised they are faster to work with. At 300dpi and 600dpi, the average run time would be 24.0 and 30.3 seconds respectively. At 300dpi the difference does not seem to be significant. However, if wanted to raise the resolution to 600dpi, saving the files as JPG files has a huge speedup. However, I did not have time to change using JPG-files instead of PNG-files. It could be argued, not saving the images at all is worth it. A speedup is gained by just adding the information whether a candidate is a point source or not, into a table. However, I think the images are useful. Therefore, they are kept in the pipeline.

5.2.3 The possibility of detecting fading objects

It was also tested if fading objects could be detected. However, the results of detecting fading objects are not very conclusive. I did manage to find a significant drop using PyNOTs Mag_auto, with a drop in magnitude of 0.076±0.074 from the first to the second image and 0.125±0.090 from the second to the third image and 0.201±0.089 from the first to the third image. The drop between image one and image two was not found to be significant with this method. However, the drop from the first to the third image has a significant drop. But, even with the errors, the results are not similar to the theoretical drop found with the alpha value of 1.53 found in the GCN[36]. The theoretical drop of 0.4523, 0.3148, and 0.7672 respectively, seems to be much higher than what was detected.

When trying to reuse apertures, to get more sources to establish a better baseline, a clearly systematic error was found, as seen in figure 21 in results. This error, I believe comes from the limitation of the precision used to WCS calibrate the images precise enough. The systematic error should not impact the results greatly if all the sources are much smaller than the apertures. However, as figure 42 shows, this is not the case for all the sources. Especially not for the transient. Because of this, I believe that the slight shift made the systematic error greater for some sources. This is also the reason we see a larger spread in the second and the third image. The detected drop with this method therefore resolved in the best result being between the first and the last image with a drop of 0.173±0.150. This is quite close to the result I found with the method described right before. However now with twice the error, making it barely significant.

Combining the method of reusing apertures with PSF, did lower the error. However, it also lowered the detected drop. Hence, this was not a much better method.

I believed, if applying PSF with the method of using cross-matching between images created by PyNOT, I could lower the error from the first method and still not have the systematic error as this method was free of that. It resulted in a higher spread as seen in figure 22. However, it did also show a drop in magnitude between the first and the last image of 0.243±0.072. Meaning, it was the largest drop detected as well as the errors got lower. Thus, the result with this method did show a significant drop. Despite of that, it is still quite far from the theoretical drop.

It seems odd that all the different methods show about the same results while not following the theoretical drop. However, the alpha value might not have been estimated from the light erupted as shortly after as the images taken by NOT. As seen in the introduction, where the light curves of GRBs are described, the alpha value will often vary depending on the time the light is observed. Many of the events initially have a lower alpha value. After some time, the light curve tends to break towards a higher alpha value. This would be a good explanation for why the theory does not match the observation, as the event of GRB220521A is also captured early by NOT. GRB970508 is an example of an event that does not follow the regular pattern as seen in figure 43[39]. Here the light curve starts off being flat, followed by an increase in magnitude before starting the expected fading.

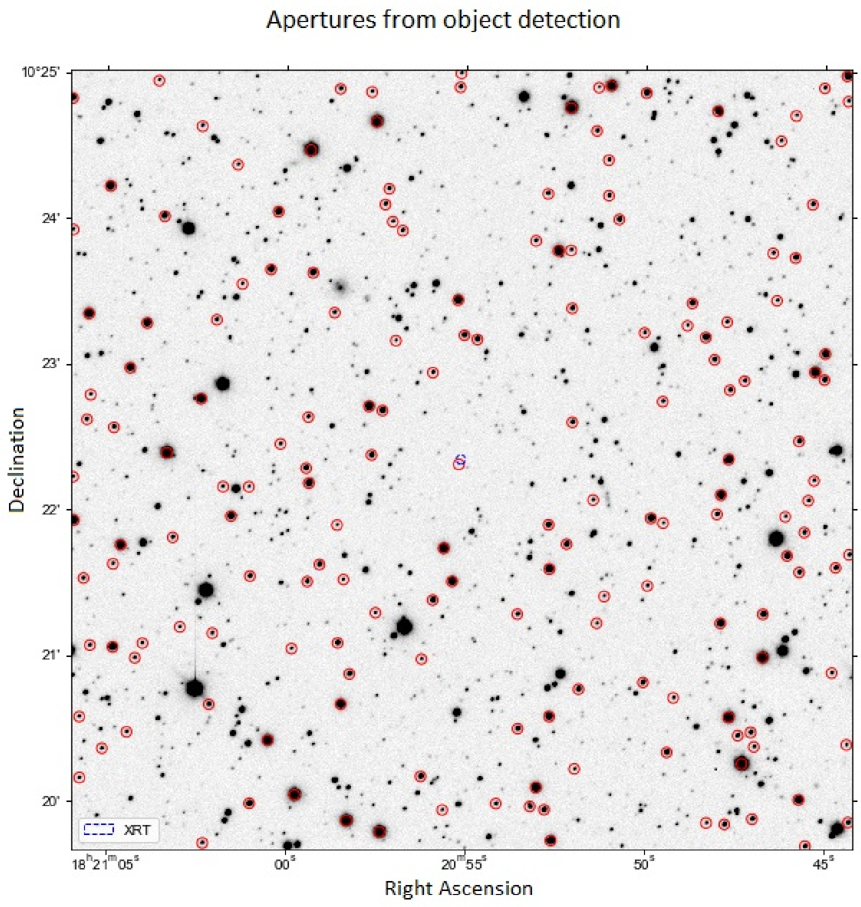


Figure 42: *Image of GRB220521A with objects detected using SEP. More sources were initially detected, but some of them were sorted out if they did not fulfill different criteria, such as how oval the source is or if it is stacked with another source or too faint.*

Trying to use fading objects for fast detection of transients is an unsuitable method for several reasons. First, the best method I developed does not seem robust as I do not get close to the theoretical drop. Moreover, the drops found if not taking the first and the last image, did barely show any significant results. Even the method I found to be the best, showed a drop in magnitude of 0.060±0.076 between the first and the second image, indicating the method is not robust. However, the method might be improved by using all the sources detected in the image instead of only using the sources that are not already found in the Gaia catalog. This means that all the sources detected would be kept as long as they are not sorted out by any of the other criterias I had for the sources. However, in a data set like the one I tested with from GRB220521A, I do not think it would make a huge difference as it already had 938, 946, and 629 unidentified sources in the images respectively. After cross-matching, it resulted in only 27 sources left. Moreover, the transient was manually forced to be kept. The reason, that the cross-matching did not work very well, was found to be because of the sources laying close. They were so close, that a decimal less in the threshold for the difference in the displacement sometimes lead to two sources, that were not the same, being saved as the same. A decimal more would sort even more sources away. Even though more sources might improve the results, I doubt it suddenly would be within the theoretical drop. And sure, it could be argued the drop is significant. Even further, let us pretend it was the case it followed the theoretical drop as well. Then the images are still taken over

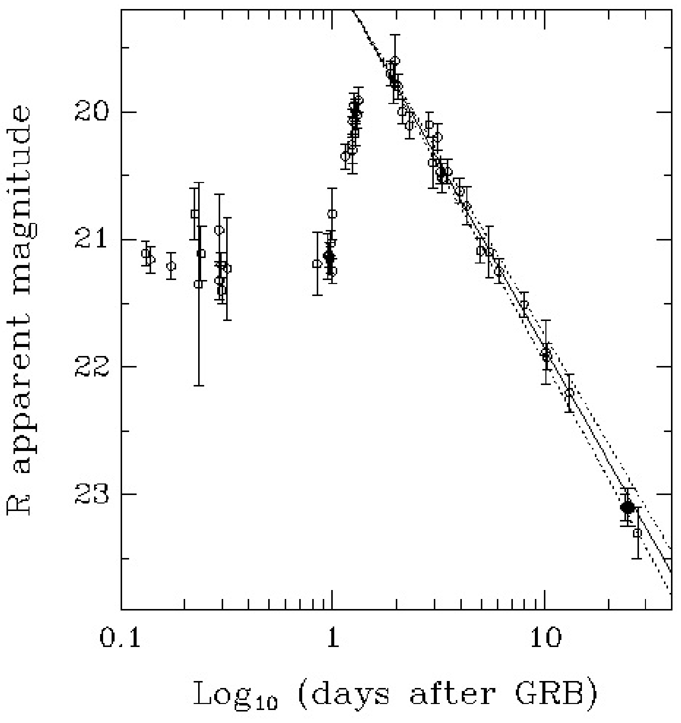


Figure 43: *This figure shows the lightcurve observed of GRB970508. The apparent magnitude of the events as a function of time is unchanged at the beginning of the event. This is followed by an increase in apparent magnitude before the light curve breaks.[39]*

a period of 14 minutes and 31 seconds. This means, transient detection in less than minutes is not possible with this method. For it to work, the images have to be taken with a much shorter exposure time. This would most likely lead to the drop in magnitude to be much less even though the drop is often larger at earlier times. As the drop detected is lower due to less time passed, the signal-to-noise ratio must be increased to get the error even further down. However, as the exposure time is less, so is the signal-to-noise ratio. Hence, the drop in magnitude would be very difficult to detect in these images. Therefore, this method is not suited for fast detection of transient objects.

5.3 Review of the Graphical user interface

The GUI created worked almost as intended. I build it such, that it should be intuitive to use by only having the most important options to choose between. However, the timing window froze when the back-end code was running. I managed to make it work as intended with simulated work by adding sleeping statements and using multiple cores running in parallel. However, using the PyNOT pipeline instead of simulated work made the GUI freeze. The reason for this was not uncovered. Besides this part not working, the rest of the GUI did. Another thing I was not quite satisfied with was the last layer displaying the information. It worked, but it had a flaw. The window does not scale to screen resolution. This is a problem for users having a lower screen resolution than 1440p(2560x1440 pixels). If the screen resolution is less, like the very common 1080p(1920x1080 pixel), the info table will be cut off from the right. Thus, the person will not be able to see the full table with information. On the other hand, if a person has a 2160p(3840 x 2160 pixels), better known as 4k, the window will not fill out the whole screen. The window can be extended to fill the screen by dragging the borders or clicking the square button in the top right corner. Though, the displayed image and corresponding table will not expand in size. It might leave it look a bit weird as it will leave empty space.

Besides the lack of scaling, the functionality is working. I am convinced a GUI is well suited for fast transient detection as I found it more intuitive than using the terminal. Further, when using multiple catalogs and several filters, the number of images becomes quite extensive and a mess to work with. Therefore, presenting the data is almost as important as the fast pipeline. For that reason, I implemented a small sorting pipeline in case someone would find the terminal better. The pipeline puts the data into folders which are sorted for catalogs and filters. However, scrolling through the images in the GUI should still be easier and faster than finding and opening each image and corresponding table one at a time.

5.4 Test of alternatives to PyNOT

Testing different tools that have the ability to do transient detection is a requisite for doing any exhortation of further work in addition to make a recommendation upon which tool is the best. For this reason, I have in addition to PyNOT, tested AutoPhotometer by Vitaly Neustroev and STDPipe by Sergey Karpov. When examining other tools than PyNOT, user experience, run time and accuracy of transient identification has to be investigated. This subsection of the discussion will be dedicated to test and discussion of AutoPhotometer and STDPipe. The test of the two tools includes installation, usability, and how well it identifies transients.

5.4.1 AutoPhotometer

I will start out with the installation of AutoPhotometer. It is written in Python, which in itself should make AutoPhotometer easy to install. However, AutoPhotometer needs SExtractor installed before installation. SExtractor is not straight forward to install on a Windows machine, as it is primarily built for Linux distributions. In the end, I had to give up on a Windows installation for SExtractor. In the light of that, you need a Linux system. I suspect a MacOS should work as well as both Linux and MacOS operating systems have plenty in common, as they are both built upon UNIX. However, I have not had the chance to test it myself. Nonetheless, after installing a Linux subsystem for Windows, more specifically Ubuntu 22.04.2 LTS, you might believe you can start installing SExtractor. Yet, before installing SExtractor, you need FFTW and ATLAS installed. Very few guides for installation are available and it is difficult to find one actually working. Meaning, a ton of searches around the internet have to be done. For that reason, I put an installation guide in Appendix for how I made it work on my computer. When FFTW, ATLAS, and SExtractor are installed, the right GPU-drivers have to be installed as well. This I made work, although it was troublesome. In all fairness, this might be because I am using a subsystem for Linux instead of just having Linux or MacOS directly installed. So, for the installation of AutoPhotometer, it can be quite difficult as it needs additional installations in preparation. Hence, it can be a lot of work for just installation.

Nonetheless, when it is installed, it is very simple to use. The images you give as input to AutoPhotometer must be reduced beforehand. For that reason, I used PyNOTs *"initialize"* and *"run_pipeline"* functions to reduce the data from GRB220101A into an image. It is supposed to be such that you can run AutoPhotometer directly from a terminal by just writing ”AutoPhotometer” followed by the file of investigation. However, this did not work. Instead, using the python-file ”autophotometer.py”, it can be run in the same manner. When downloading Autophotometer from git, you get two example files of which you can test AutoPhotometer. I did not manage to make it work with the flag ”-ac” referred to in the README-file[40]. However, this is for uncorrected images. The file I used was corrected, so it should not be a problem. For the corrected example image, Authophotomer works as I believe it is supposed to. However, when I test it on the reduced image from GRB220101A, I get a narrow plot where it seems like there is nothing. It turns out, that zooming in reveals that AutoPhotometer has done something. However, it seems like something is wrong in the resulting plot. Further, it was found that the zero point calibration was wrong. This might be the reason for the resulting image also appearing to be wrong. However, it did provide a histogram of FWHM, which looked as expected and somewhat similar to what I have got myself for the same image.

As it might have been noticed, I have no plots from AutoPhotometer here. This was due to a crash in my Ubuntu. I tried reinstalling it again. However, there was a driver issue I could not correct. As already mentioned, I believe the driver issue might not be a problem on a MacOS or Linux, as it was a specific error related to my Nvidia GPU communicating through my subsystem. To sum it up, I was not able to use AutoPhotometer for transient detection as I did not manage to get any useful results. On top of that, it was very difficult to install.

5.4.2 STDPipe

For the installation of STDPipe, SExtractor must be installed. So, the difficulties for that part are the same as when I installed AutoPhotometer. However, STDPipe does write in their README file that it uses SExtractor or SEP for object detection. Indicating SEP could be used instead. SEP is the library version of SExtractor, written in Python. Meaning, it is easy to install, as it can be installed through conda or pip [41]. On that note, as I had already installed SExtractor, this is what I used. Therefore, I do not know the workload of changing their example code to use SEP. Besides SExtractor, another program named HOTPANTS has to be installed. This program is used to detect changes, such as fading of an object between images. If working, it could be a strong feature. Sadly, I did not manage to make it work. After spending a lot of time, it seemed like I got HOTPANTS compiled correctly. However, it still did not work when I tried using it in a script. How I installed it is included in the appendix in the guide for installing SExtractor, ATLAS, FFTW, AutoPhotmeter, STDPipe, and HOTPANTS.

When it comes to the usage of STDPipe, there is a STDPipe tutorial notebook. It is in the GitHub you clone before installation. The tutorial is a simple pipeline with examples of what to expect of results and a pipeline for transient detection. Again, I used the reduced image from GRB220101A created with PyNOT. It was very easy to change the path to the location of my image from GRB220101A in the already existing tutorial together with some other very minor changes. Afterward, it was easy to run the notebook. I did not manage to find the transient though.

In summary, I had trouble with the installation of HOTPANTS, which makes it difficult to judge the full capabilities of the STDPipe. However, it is used by those working on the SWOM(Space-based multi-band astronomical Variable Objects Monitor)-mission as well as I can see it is updated in the

GitHub on a regular basis. Therefore, it might be interesting to make a deeper investigation of the STDPipe. This can maybe be done in cooperation with the team working with the SWOM mission as they already use it for their satellite. Because of that, they might have some good tips and tricks to get good results. This, I believe even though I did not myself got useful results.

5.4.3 Small summary of alternatives

Even though the work on AutoPhotometer is great, I do not recommend going further with it. Mostly based on two major points. First, it relies on other programs to be installed besides Python. The dependence on SExtractor limits the use of AutoPhotometer across platforms. It is to cope if the source code is changed to use SEP instead of SExtractor. Secondly, the transient detection did not only work for the example image and not for GRB220101A. A description of requirements for the reduced image as input is not in the documentation. Thus, I can not figure out if it might be something with the image I created through PyNOT, its header or an error in AutoPhotometer. The zero point was estimated wrong, and it might be, if AutoPhotometer could take zero-point as an optional input like PyNOT, it would solve this problem. However, this is mostly my guess, and it might very well be wrong.

I claimed it could be interesting to look deeper into STDPipe. Nonetheless, I am for now, convinced the time is better spent making further work on PyNOT. I believe this for two reasons. First of all, STDPipe, like AutoPhotometer, is dependent on other programs to be installed. With STDPipe, I do not see the problem being coped with. Unless someone knows how to install HOTPANTS. If not, I think the feature of looking for fading objects is a very strong feature missing. Therefore, I do not see the large advantage of using the pipeline for transient detection compared with PyNOT. Moreover, it is not even sure it would be possible with HOTPANTS to detect fading between images. The images would have to be taken with short exposure rapidly after each other for fast transient detection as discussed earlier. Secondly, I did not manage to get any good results out when using their example pipeline. Because of those reasons, my final recommendation is not to go further with the STDPipe.

I found that PyNOT is a stronger case. It works across platforms, is easy to use and the results come in a format easier to review. Further, it has the full pipeline from raw images into transient detection. So, it has no need for another program to reduce the image. As a result, I believe neither AutoPhotometer nor STDPipe is where the time should be spent. Instead, I think the time should be spent developing further on PyNOT.

5.5 Work for future

For future work, I will recommend doing further development of PyNOT. I have tested both AutoPhotometer and STDPipe beside PyNOT and I found PyNOT to be easier to install and easier to use as well as it delivered the best results. Moreover, it is somewhat easy to create further scripts upon as PyNOT is very simple built. Its only method for transient detection is comparing with catalogs and having error circles for BAT and XRT coordinates. This method was found to be strong. However, I wonder why the threshold of angular distance between objects in the catalog and objects from object detection had to be changed for GRB220521A. This is indicating something might be wrong that must be corrected and that it is in need of further investigation.

Moreover, the automatic calibration of zero point must be improved to work more consistently. It was not robust enough to be used for consistent fast transient detection. A proposal for a solution could be letting PyNOT try finding a zero point. And in the cases where it does not find a zero point, the script should wait for the user to manually give an input for the zero point in the terminal. Another patch solution could be, if PyNOT does not find a zero point, using 27.5 as default. This might not be the best solution and will give a slight shift in the magnitudes detected. Nonetheless, it is still a better solution as it does not waste time having to run the pipeline again. This is important when locating the position of the transient fast is a priority.

This leads to another minor inconvenience of PyNOT. Fixing PyNOT to overwrite existing data instead of having to manually delete it when running the pipeline again would make the workflow of PyNOT smoother. This is something some might do if, for example, some would adjust the zero point, change the magnitude threshold or maybe even change the threshold for angular distance in the source code.

Further, changing the PyNOT pipeline, such as it would only put in squares if an object were detected inside, would be an improvement. Using catalogs like the WISE in optical images will leave many squares indicating an object is there, even though there is none in the optical image. This is just distracting.

PyNOT could also be updated such it can use any other arbitrary catalog to compare against. This could be changed such that it automatically queried based on an input from the user in the $"findnew"$ function. If no input was given it should then use Gaia as default. This would make PyNOT usable in more bands and thereby also suitable for the upcoming NTE.

Further, I would implement a method for finding point sources. This could very well be with the method I have used in this report using FWHM. This proved to be quite effective and would be a strong feature.

As PyNOT only is using one method for transient detection at the moment, it has the potential to be improved further. It has a strong fundamental as it is simple build, but also robust.

If a graphical user interface should be built to make use of PyNOT and extensions it would make the data easier to comprehend. Something like the already existing SAOImageDS9 would be a good place for inspiration as it is easy to use and has many other good functions for deeper investigations as well. PyNOT would in fact be a good supplement for SAOImageDS9.

6

CONCLUSION

Data of the late stages of the afterglows of some of the most luminous events called gamma-ray bursts is becoming plentiful. In the meantime, data of the early stages are lacking and even worse is that the data of the prompt emission is very scarce. The lacking data is mostly due to the lack of automatic tools for fast and robust transient detection as those events are almost as brief as they are luminous. In this report, I have thoroughly tested PyNOT and its current capabilities in order to determine if PyNOT is the best fitted tool to be used at the Nordic Optical Telescope for very fast detection of transient events. PyNOT was tested with that in mind that the prompt emission of GRBs most often is gone within minutes. Additionally, it was explored which methods PyNOT was lacking that could be useful to implement.

PyNOT is currently using two methods for transient detection. PyNOT is creating error circles linked to the errors of the coordinates provided by BAT and XRT onboard the SWIFT satellite as well as PyNOT compare objects detected in an image against sources in the Gaia catalogue. Those methods were found to be very strong and very robust for transient detection if it was not for the flaw that sources would be completely ignored if they are fainter than a threshold for magnitude set in PyNOT. This problem should preferably be corrected in PyNOT. Otherwise, PyNOT can still be used if a higher threshold is chosen. One of PyNOTs features is that it automatically can zero point calibrate but this feature was neither found to be robust enough to use as it is at the moment. Besides this, PyNOT is found to be an easy to use tool, that can take raw data directly from the Nordic Optical Telescope and process it and do transient detection in times less than a minute for most data sets.

In this project, it was found that PyNOT would benefit if a method for selecting point sources was implemented. The method and pipeline implemented in this report were found to be both robust and efficient. Moreover, it was found that implementing the possibility to use several arbitrary catalogues instead of only the Gaia catalogue would be a good feature as it would make PyNOT more versatile. Using arbitrary catalogues will be especially beneficial for the upcoming NOT Transient Explorer and other future instruments at NOT. Trying to implement a tool in PyNOT that can detect fading objects was not found to be recommended for fast transient detection. It was also found a GUI would make it easier to use PyNOT, especially when it comes to reviewing the produced results if more catalogues are used with several filters.

Considering how PyNOT did compared against AutoPhotometer and STDPipe my recommendation is to do further work on PyNOT. It provides better results as well as PyNOT is easier to use. The fact that you do not have to manually do data processing of the images beforehand using PyNOT as well as the results produced are gathered into one figure and one table for each filter makes PyNOT very strong for fast transient detection. If implementing a point source selection pipeline into PyNOT, it should be possible with the PyNOT pipeline to detect and locate a transient event quite robust in less than a few minutes after BAT is triggered. If some would wait up to an additional 100 seconds for the XRT coordinates, the GRBs location can be determined with very high confidence through the PyNOT pipeline. In conclusion, PyNOT has room for some improvements, but it is a strong tool with a great potential to catch GRBs early on if slightly upgraded.

REFERENCES

[1] NASA Content Administrator. *Oldest Recorded Supernova*. URL: https://www.nasa.gov/multimedia/imagegallery/image_feature_2173.html. (accessed: 24.11.2022).

[2] Gilbert Vedrenne and Jean-Luc Atteia. *Gamma-ray bursts - The brightest explosions in the universe, s. 1*. Springer, 2009. ISBN: 978-3-540-39085-5.

[3] U.S. Department of State. *Treaty Banning Nuclear Weapon Tests in the Atmosphere, in Outer Space, and Under Water*. URL: https://2009-2017.state.gov/t/avc/trty/199116.htm. (accessed: 22.11.2022).

[4] Peter Schneider. *Extragalactic Astronomy and Cosmology an introduction, s. 402-405*. Springer, 2006. ISBN: 3-540-33174-3.

[5] Carl Akerlof. *Observation of contemporaneous optical radiation from a -ray burst*. URL: https://arxiv.org/pdf/astro-ph/9903271.pdf. (accessed: 28.11.2022).

[6] Asaf Pe'er and Felix Ryde. *Observations, theory and implications of thermal emission from gamma-ray bursts*. URL: https://arxiv.org/pdf/1003.2582.pdf. (accessed: 28.11.2022).

[7] S. Schanne et al. "The ECLAIRs micro-satellite for multi-wavelength studies of gamma-ray burst prompt emission" (2006). URL: https://www.researchgate.net/publication/3139607_The_ECLAIRs_micro-satellite_for_multi-wavelength_studies_of_gamma-ray_burst_prompt_emission.

[8] Pawan Kumar and Bing Zhang. "The Physics of Gamma-Ray Bursts Relativistic Jets" (2014). URL: https://arxiv.org/pdf/1410.0679.pdf.

[9] E. Ramirez-Ruiz N. Gehrels and D. B. Fox. "Gamma-Ray Bursts in the Swift Era" (2009). URL: https://arxiv.org/pdf/0909.1531.pdf.

[10] Gilbert Vedrenne and Jean-Luc Atteia. *Gamma-ray bursts - The brightest explosions in the universe, Chapter 8. s. 385-462*. Springer, 2009. ISBN: 978-3-540-39085-5.

[11] NBI. *Extremely energetic particles coupled with the violent death of a star for the first time*. URL: https://www.nbi.ku.dk/english/news/news19/extremely-energetic-particles-coupled-with-the-violent-death-of-a-star-for-the-first-time/. (accessed: 07.03.2023).

[12] Gilbert Vedrenne and Jean-Luc Atteia. *Gamma-ray bursts - The brightest explosions in the universe, Chapter 5. s. 219-252*. Springer, 2009. ISBN: 978-3-540-39085-5.

[13] D. A. Kann etal. "THE AFTERGLOWS OF SWIFT-ERA GAMMA-RAY BURSTS. I. COMPARING PRE-SWIFT AND SWIFT ERA LONG/SOFT (TYPE II) GRB OPTICAL AFTERGLOWS" (2010). URL: https://arxiv.org/pdf/0712.2186.pdf.

[14] NASA. *GCN: The Gamma-ray Coordinates Network (TAN: Transient Astronomy Network)*. URL: https://gcn.gsfc.nasa.gov/. (accessed: 13.08.2023).

[15] Gilbert Vedrenne and Jean-Luc Atteia. *Gamma-ray bursts - The brightest explosions in the universe, s. 160-165*. Springer, 2009. ISBN: 978-3-540-39085-5.

[16] NASA. *Overview of the Fermi GBM*. URL: https://fermi.gsfc.nasa.gov/ssc/data/analysis/documentation/Cicerone/Cicerone_Introduction/GBM_overview.html. (accessed: 13.08.2023).

[17] Jens-Kristian Krogager. *PyNOT-redux*. URL: https://pypi.org/project/PyNOT-redux/. (accessed: 21.11.2022).

[18] Peter Schneider. *Extragalactic Astronomy and Cosmology an introduction, s. 35-36*. Springer, 2006. ISBN: 3-540-33174-3.

[19] International Earth Rotation and Reference Systems Service. *The International Celestial Reference System (ICRS)*. URL: https://www.iers.org/IERS/EN/Science/ICRS/ICRS.html. (accessed: 13.08.2023).

[20] Arne Ardeberg. *SOME PROPERTIES of the NORDIC OPTICAL TELESCOPE*. URL: https://www.not.iac.es/telescope/tti/proptxt.html. (accessed: 13.08.2023).

[21] Peter Schneider. *Extragalactic Astronomy and Cosmology an introduction, s. 40*. Springer, 2006. ISBN: 3-540-33174-3.

[22] ESA. *Expected science performance for the nominal and the extended mission based on Gaia (E)DR3*. URL: https://www.cosmos.esa.int/web/gaia/science-performance. (accessed: 16.07.2023).

[23] Maia Nenkova Zeljko Ivezic Robert Nikutta Nicholas Hunt-Walker and Moshe Elitzur. *The meaning of WISE colours – I. The Galaxy and its satellites*. URL: http://faculty.washington.edu/ivezic/Publications/WISE1.pdf. (accessed: 16.07.2023).

[24] Peter Schneider. *Extragalactic Astronomy and Cosmology an introduction, s. 25*. Springer, 2006. ISBN: 3-540-33174-3.

[25] Peter Schneider. *Extragalactic Astronomy and Cosmology an introduction, s. 23*. Springer, 2006. ISBN: 3-540-33174-3.

[26] GCN. *Object close to GRB220101A*. URL: https://gcn.gsfc.nasa.gov/gcn3/31373.gcn3. (accessed: 08.06.2023).

[27] Stefan Czesla. *Source code for angular distance at PyAstronomy GitHub*. URL: https://github.com/sczesla/PyAstronomy/blob/master/src/pyasl/asl/angularDistance.py. (accessed: 13.08.2023).

[28] Neil O'Mahony. *Results from the Calibration of ING's RoboDIMM*. URL: https://www.ing.iac.es/astronomy/development/seeing/Correl_DIMM.html. (accessed: 14.07.2023).

[29] Peter Schneider. *Extragalactic Astronomy and Cosmology an introduction, s. 98*. Springer, 2006. ISBN: 3-540-33174-3.

[30] Michael Hilker Steffen Mieske and Leopoldo Infante. "Fornax compact object survey FCOS: On the nature of Ultra Compact Dwarf galaxies" (2004). URL: https://arxiv.org/pdf/astro-ph/0401610.pdf.

[31] L. Yang et al. "Early results from GLASS-JWST. V: the first rest-frame optical size-luminosity relation of galaxies at" (2022). URL: https://arxiv.org/pdf/2207.13101.pdf.

[32] Peter Schneider. *Extragalactic Astronomy and Cosmology an introduction, s. 155*. Springer, 2006. ISBN: 3-540-33174-3.

[33] Tim Ruscica. *TechWithTim PyQt5 tutorial on youtube*. URL: https://www.youtube.com/watch?v=FVpho_UiDAY&list=PLzMcBGfZo4-lB8MZfHPLTEHO9zJDDLpYj&index=3. (accessed: 08.06.2023).

[34] NumPy Developers. *What is NumPy?* URL: https://numpy.org/doc/stable/user/whatisnumpy.html. (accessed: 09.08.2023).

[35] Christa Gall Marianne Vestergaard Lise Christensen Luca Izzo. *Exercise 2: Basic Processing of Imaging Data, Astronomical Data Processing Course, 2022, Copenhagen University*. URL: https://absalon.ku.dk/courses/61398/files/folder/Exercise%20sheets%20Python?preview=6329131. (accessed: 14.08.2023).

[36] *GCN 220521A*. URL: https://gcn.gsfc.nasa.gov/other/220521A.gcn3. (accessed: 28.06.2023).

[37] NOT. *ALFOSC Zero point monitoring*. URL: https://www.not.iac.es/instruments/alfosc/zpmon/. (accessed: 12.08.2023).

[38] ESA. *Euclid overview*. URL: https://www.esa.int/Science_Exploration/Space_Science/Euclid_overview. (accessed: 12.08.2023).

[39] Pian etal. “Hubble Space Telescope Imaging of the Optical Transient Associated with GRB970508” (1997). URL: https://arxiv.org/pdf/astro-ph/9710334.pdf.

[40] Vitaly Neustroev. *AutoPhotometer GitHub*. URL: https://github.com/VitalyAstro/autophotometer/blob/main/README.md. (accessed: 18.07.2023).

[41] Kyle Barbary. *SEP documentation*. URL: https://sep.readthedocs.io/_/downloads/en/stable/pdf/. (accessed: 15.06.2023).

[42] Peter Schneider. *Extragalactic Astronomy and Cosmology an introduction, s. 159*. Springer, 2006. ISBN: 3-540-33174-3.

[43] S.Nick. *Resizing table widget when window is maximized*. URL: https://stackoverflow.com/questions/50890645/resizing-table-widget-when-window-is-maximized. (accessed: 12.08.2023).

[44] Neamerjell. *How to search and replace text in a file?* URL: https://stackoverflow.com/questions/17140886/how-to-search-and-replace-text-in-a-file. (accessed: 14.08.2023).

[45] Photutils Developers. *PSF Photometry*. URL: https://photutils.readthedocs.io/en/stable/psf.html. (accessed: 12.08.2023).

Appendices

APPENDIX OVERVIEW

1. The results from the transient detection with PyNOT for GRB190114A, GRB190114B, and GRB211211A. The results are with the r-filter and using the Gaia catalog. [2 pages]

2. The results from the point source selection pipeline for GRB190114A, GRB190114B, GRB190114C, and GRB211211A. The results are from the r-filter and using the Gaia catalog. [2 pages]

3. Result of GRB220101A with the use of the WISE catalog. [1 page]

4. BIAS file for GRB190114A [1 page]

5. Script for the cosmological model. To help with the model I have used source [42] and [32] [1 page]

6. The second script is for running the GUI. In the script I am referencing to [43] [7 pages]

7. The third script is the extended PyNOT pipeline used in the GUI. In the script I am referencing to [44] [3 pages]

8. The fourth script is the sorting pipeline used in the GUI [1 page]

9. The fifth script is the cross-matching pipeline between catalogs used in the GUI [3 pages]

10. The sixth script is creating the ”most important info table” used in the GUI [1 page]

11. The seventh script is the point source selection pipeline used in the GUI [6 pages]

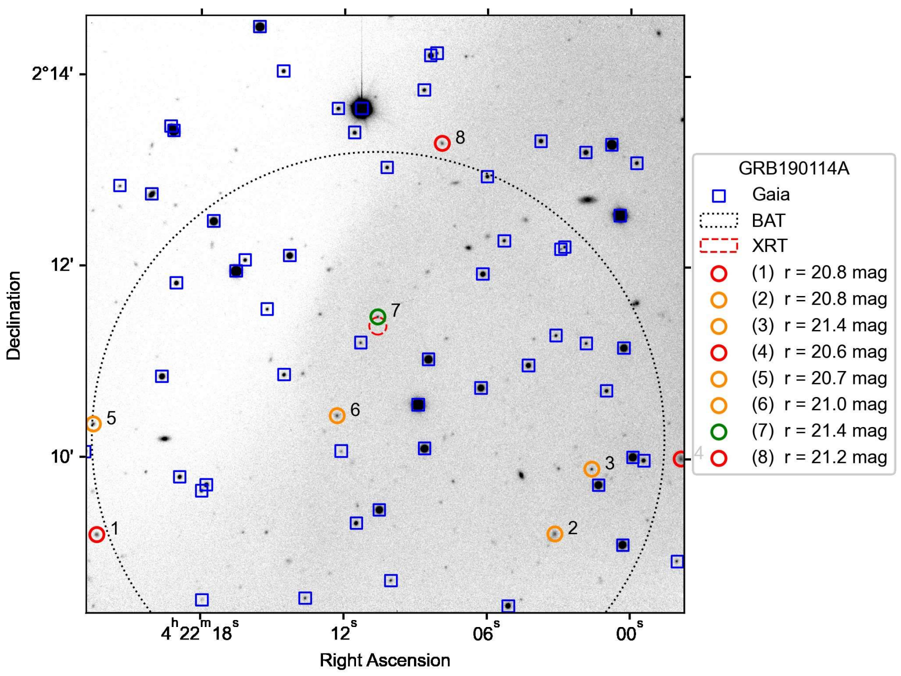
12. The eighth script is the ”MainWindow” in the GUI [5 pages]

13. The ninth script is the test of PyNOT. It is shown for the test of galactic height, and the code is very similar to the code for right ascension [5 pages]

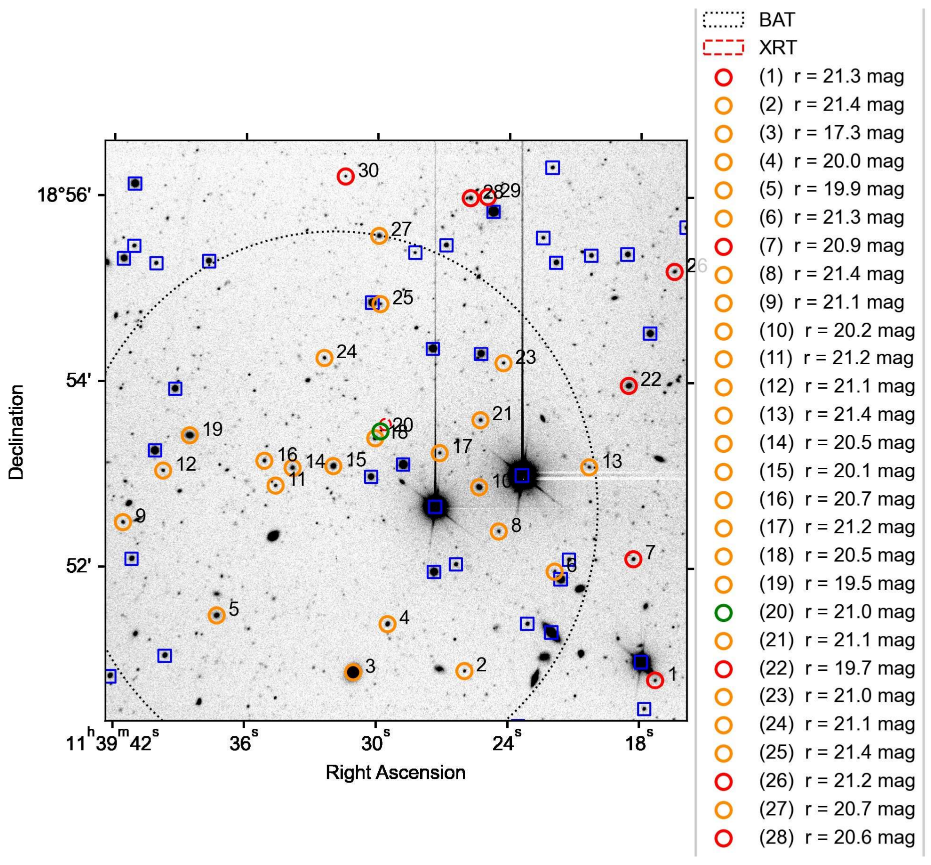
14. The tenth script is for testing if fading objects are possible. In the script I am referencing to [45] [10 pages]

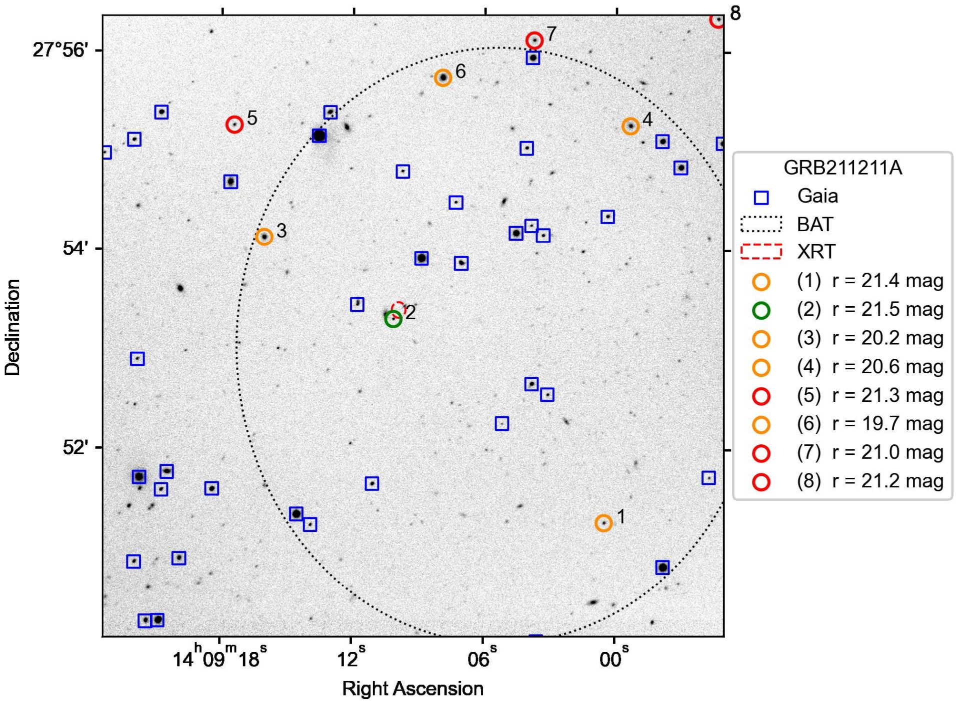
15. Small guide to installation of FFTW, ATLAS, SeXtractor, AutoPhotometer, STDPipe and HOT-PANTS [1 page]

Results from the point transient detection:



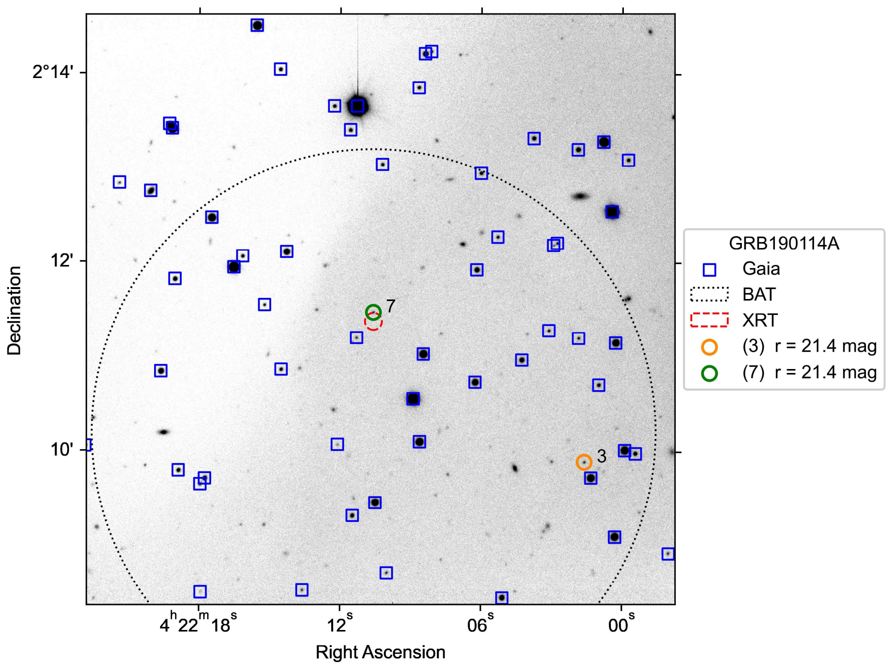
GRB190114B:

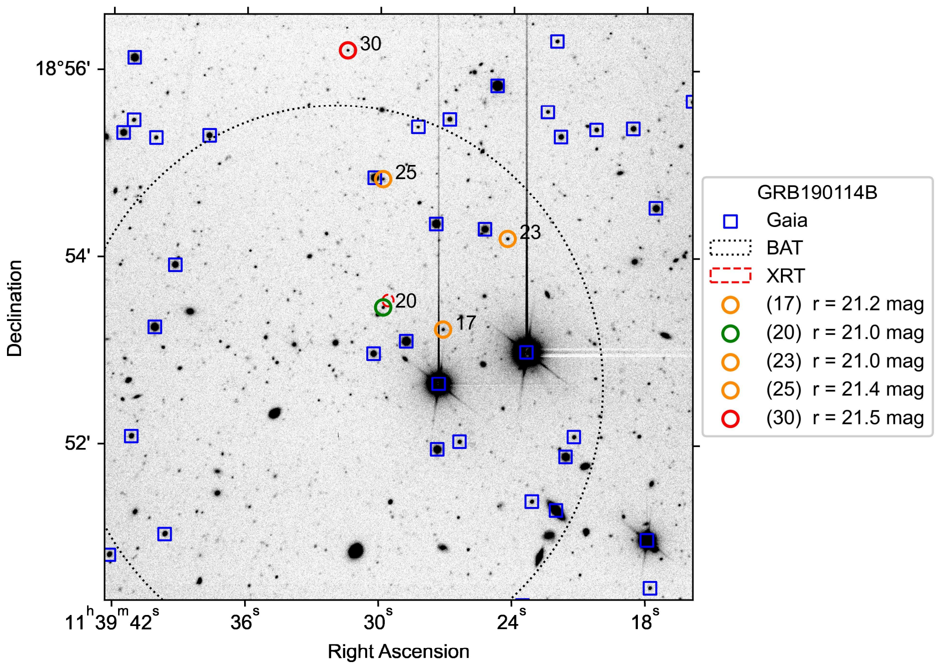


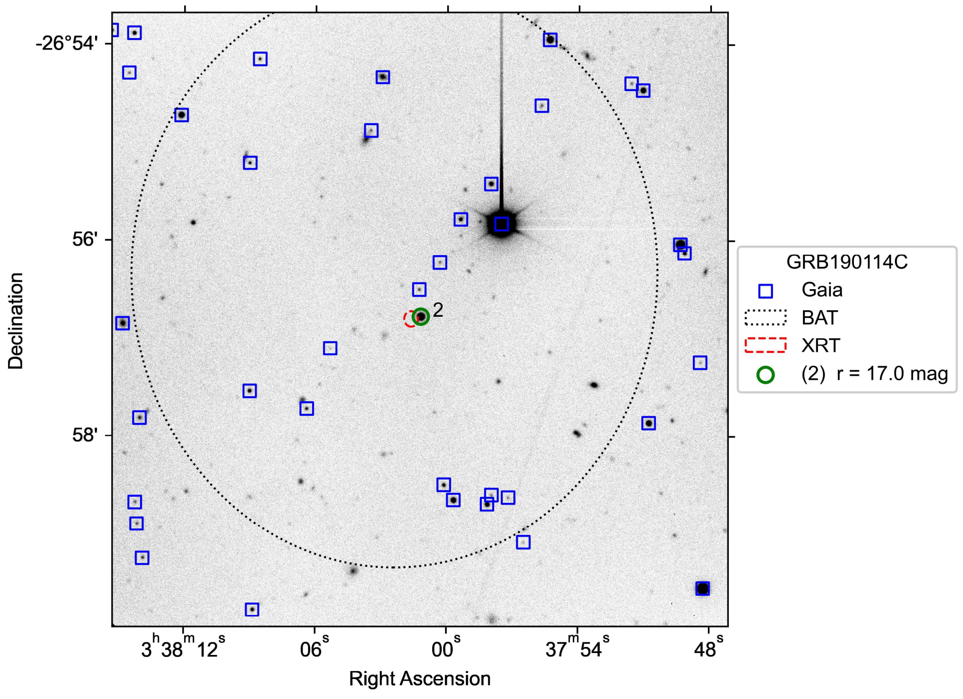


27°56′
54′
52′
Declination
14h09m18s
12s
06s
00s
Right Ascension
1
2
3
4
5
6
7
8
GRB211211A
Gaia
BAT
XRT
(1) r = 21.4 mag
(2) r = 21.5 mag
(3) r = 20.2 mag
(4) r = 20.6 mag
(5) r = 21.3 mag
(6) r = 19.7 mag
(7) r = 21.0 mag
(8) r = 21.2 mag

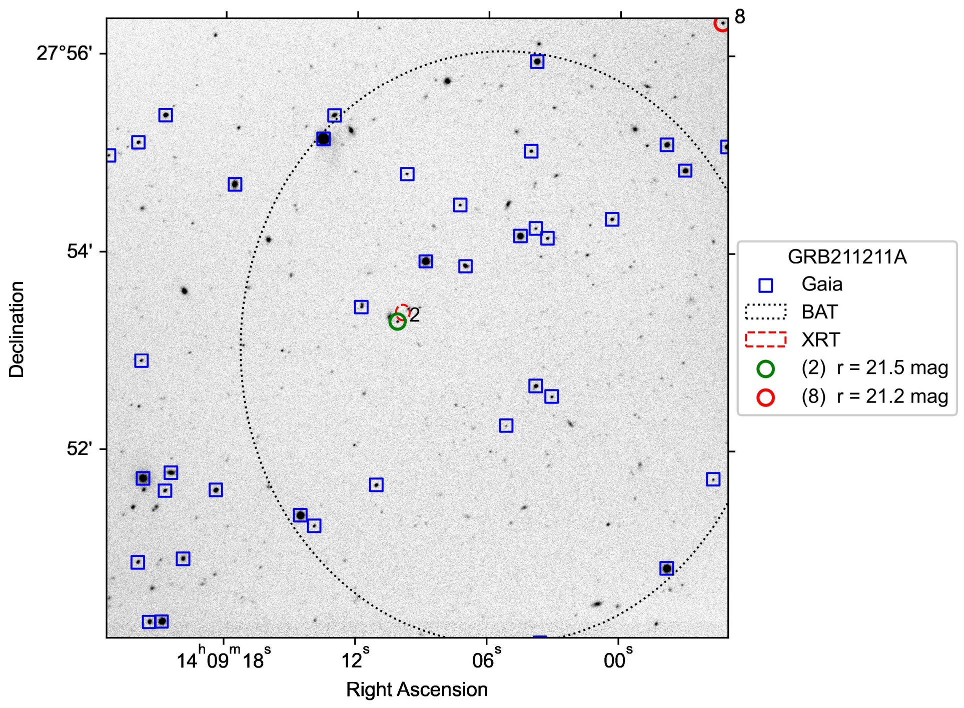
Results from the point source selection pipeline:



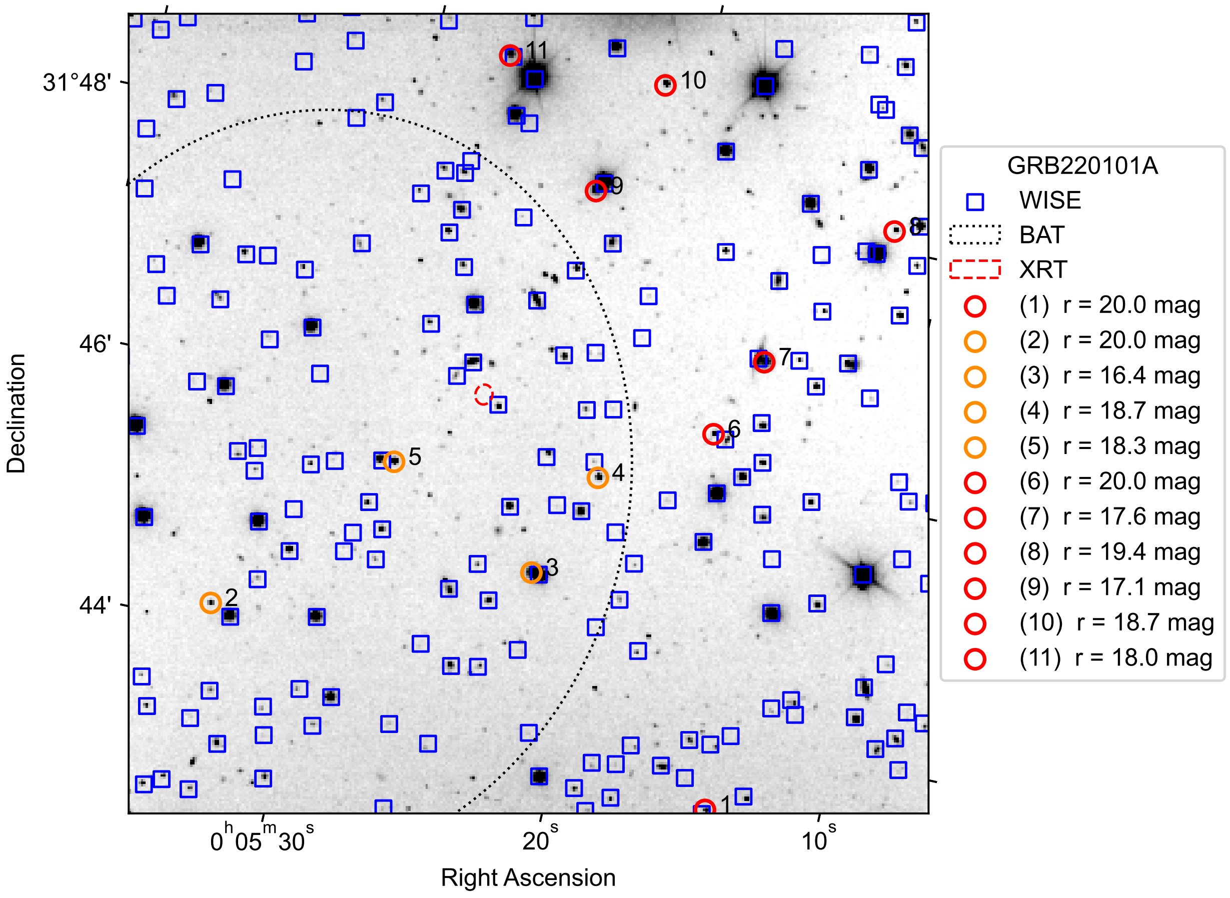




-26°54'
56'
58'
Declination
3h38m12s
06s
00s
37m54s
48s
Right Ascension
2
GRB190114C
Gaia
BAT
XRT
(2) r = 17.0 mag

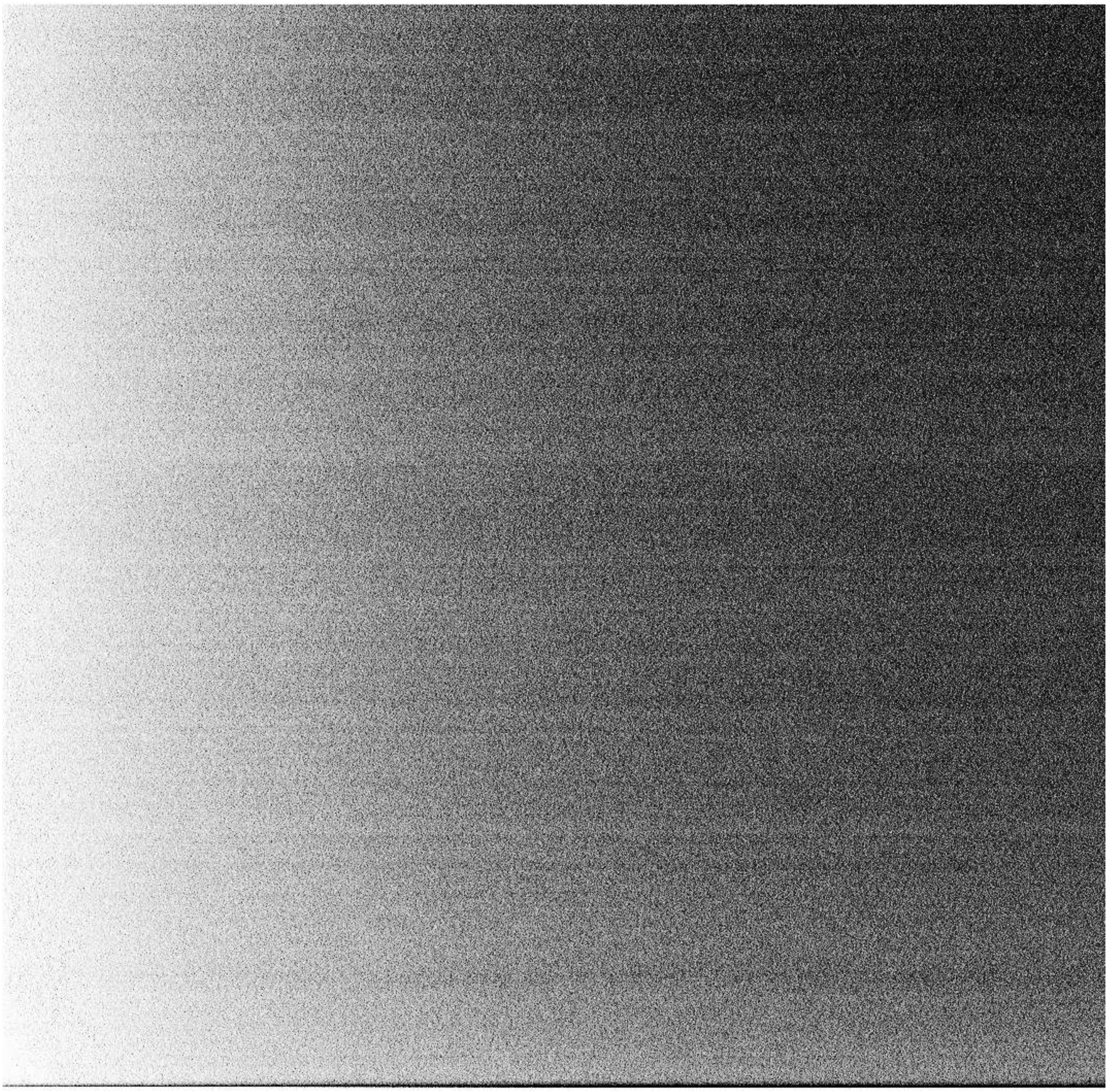


27°56'
54'
52'
Declination
14h09m18s
12s
06s
00s
Right Ascension
2
8
GRB211211A
Gaia
BAT
XRT
(2) r = 21.5 mag
(8) r = 21.2 mag



31°48'
46'
44'
Declination
0h05m30s
20s
10s
Right Ascension
1
2
3
4
5
6
7
8
9
10
11
GRB220101A
WISE
BAT
XRT
(1) r = 20.0 mag
(2) r = 20.0 mag
(3) r = 16.4 mag
(4) r = 18.7 mag
(5) r = 18.3 mag
(6) r = 20.0 mag
(7) r = 17.6 mag
(8) r = 19.4 mag
(9) r = 17.1 mag
(10) r = 18.7 mag
(11) r = 18.0 mag

BIAS file for GRB190114A



```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as spi
from numpy import sqrt, sin, sinh


def f(x):
    if kappa == 0:
        return x
    if kappa > 0:
        return 1/sqrt(kappa) * sin(sqrt(kappa)*x)
    if kappa < 0:
        return 1/sqrt(-kappa) * sinh(sqrt(-kappa)*x)

def x(z):
    def integration_func(a):
        return (c/H) /sqrt(a*Omega_m + a**2*(1-Omega_m-Omega_L)+ a**4*Omega_L)
    I = spi.quad(integration_func,(1+z)**(-1),1)
    return I[0]


def AngDist(z):
   return f(x(z))/(1+z)


c = 2998.
H = 0.7
Omega_L = 0.7
Omega_m = 0.3

cH = c/H
kappa = (Omega_m+Omega_L-1.)/cH**2.

z = np.arange(0,15,0.1)
sizes = [AngDist(i)*2.*np.pi/(3600.*360.)*1000. *0.6 for i in z]

plt.figure(figsize = (8, 4))
plt.plot(z,sizes)
plt.title("Proper size of object filling 0.6 arcseconds as function of z")
plt.ylabel('Size (kpc/arcsec)')
plt.xlabel('Redshift')
plt.show()
plt.savefig("cdm-model.jpg", dpi = 500)
```

```python
#%%
import os
import sys
import time
import pandas as pd
import matplotlib.pyplot as plt

from PyQt5 import uic, QtWidgets, QtCore, QtGui
from PyQt5.QtCore import QObject, QThread, pyqtSignal
from PyQt5.QtWidgets import QFileDialog, QPushButton, QWidget, QMessageBox

from MainWindow import Ui_MainWindow
from PyNOT_ext_pipeline import full_pipeline
from SortingPipeline import SortPipeline
from PointsourcePipeline import PointsAllCatsNFilters
from CrossmatchingPipeline import crossmatch
from ImportantInfoTablePipeline import ImportantInfoTable


class Ui_FrontWindow(QtWidgets.QMainWindow):
    def __init__(self):
        super(Ui_FrontWindow, self).__init__()
        uic.loadUi('frontpage_gridlayout.ui', self)

        #Varibles
        self.path = ""
        self.batRA = ""
        self.batDEC = ""
        self.batERROR = ""
        self.xrtRA = ""
        self.xrtDEC = ""
        self.xrtERROR = ""
        self.ZP_input = ""


        #Function connections
        self.browsefoldersButton_2.clicked.connect(self.get_directory)

        self.batra_input.textChanged.connect(self.batRA_input)
        self.batdec_input.textChanged.connect(self.batDEC_input)
        self.baterror_input.textChanged.connect(self.batERROR_input)

        self.xrtra_input.textChanged.connect(self.xrtRA_input)
        self.xrtdec_input.textChanged.connect(self.xrtDEC_input)
        self.xrterror_input.textChanged.connect(self.xrtERROR_input)

        self.zp_input.textChanged.connect(self.ZPInput)

        self.submit_pushButton.clicked.connect(self.Submit)

        self.show()

        self.test()

    def test(self):
        print("Start")

    def get_directory(self):
        path_folder = QFileDialog.getExistingDirectory(self)
        path_folder_splitted = path_folder.split("/")
        self.yourfolder_label.setText("""Your folder: " + f"../../{path_folder
                                      _splitted[-3]}/{path_folder_splitted[-2]}
                                      /{path_folder_splitted[-1]}""")
        self.path = path_folder + "/"


    def batRA_input(self, text):
        if text.replace('.','',1).isdigit():
            self.batRA = float(text)
        else:
            self.batRA = ""

    def batDEC_input(self, text):
        if text.replace('.','',1).isdigit():
            self.batDEC = float(text)
        else:
            self.batDEC = ""

    def batERROR_input(self, text):
        if text.replace('.','',1).isdigit():
            self.batERROR = float(text)
        else:
            self.batERROR = ""

    def xrtRA_input(self, text):
        if text.replace('.','',1).isdigit():
            self.xrtRA = float(text)
        else:
            self.xrtRA = ""

    def xrtDEC_input(self, text):
        if text.replace('.','',1).isdigit():
            self.xrtDEC = float(text)
        else:
            self.xrtDEC = ""

    def xrtERROR_input(self, text):
        if text.replace('.','',1).isdigit():
            self.xrtERROR = float(text)
        else:
            self.xrtERROR = ""

    def ZPInput(self, text):
        if text.replace('.','',1).isdigit():
            self.ZP_input = float(text)
        else:
            self.ZP_input = ""

    def submit(self):
        print(self.batRA)
        print(self.batDEC)
        print(self.batERROR)
        print(self.xrtRA)
        print(self.xrtDEC)
        print(self.xrtERROR)
        print(self.ZP_input)



    def Submit(self):
        # if PyNOT already ran checked:
```

```python
if (self.path and
    self.ranpynotcheckBox.isChecked() == True):
    print("Checked")

    #Go to next page
    #Open infowindow
    infowindow.setupUi(InfoWindow, self.path)
    InfoWindow.show()

#Checks if a path to the folder has been choosen and Coordinates
# - for BAT or XRT is given.
elif (self.path and
        self.batRA and self.batDEC and self.batERROR
        or self.xrtRA and self.xrtDEC and self.xrtERROR):



    self.DoesWorkWidget = Ui_DoesWorkWidget(self.path,
                                            self.batRA,
                                            self.batDEC,
                                            self.batERROR,
                                            self.xrtRA,
                                            self.xrtDEC,
                                            self.xrtERROR,
                                            self.ZP_input)
    self.DoesWorkWidget.show()
    self.close()


else:
    #Make popup telling it need a path or coordinates:
    if not(self.path):
        return_message = "You have to choose a path!"
        detailed_message = ("""If you do not choose a path the program
                            want know where it should work. """
                            "\n \n"
                            "This would make the program go sad sad. "
                            "\n \n"
                            """Please choose the folder containing the
                            folder with the raw images of the GRB you
                            want to observe.""")
    else:
        return_message = """You have to give at coordinates with error
                            for at least BAT or XRT!"""
        detailed_message = ("""You have to give at coordinates with
                            error for at least BAT or XRT."""
                            "\n \n"
                            """The coordinates have to be in degrees
                            be in degrees."""
                            "\n \n"
                            """Float number are seperated with a
                            punctuation '.' NOT a comma ','""")

    msg = QMessageBox()
    msg.setWindowTitle("messagebox")
    msg.setText(return_message)
    msg.setIcon(QMessageBox.Information)
    msg.setDetailedText(detailed_message)
    x = msg.exec_()
```

```python
#%%

class Ui_DoesWorkWidget(QtWidgets.QMainWindow):
    def __init__(self, path, batRA, batDEC, batERROR, xrtRA, xrtDEC, xrtERROR,
                                                                          ZP_input):
        super(Ui_DoesWorkWidget, self).__init__()
        uic.loadUi('DoesWorkLayer.ui', self)
        self.show()


        self.application_path = os.getcwd()
        #Varibles
        self.path = path
        self.batRA = batRA
        self.batDEC = batDEC
        self.batERROR = batERROR
        self.xrtRA = xrtRA
        self.xrtDEC = xrtDEC
        self.xrtERROR = xrtERROR
        self.ZP_input = ZP_input

        self.p_pynot = True
        self.p_sort = True
        self.p_points = True
        self.p_cross = True
        self.p_tabs = True

        self.start_time = time.time()
        #self.runTimer()
        #self.runProcessing()
        self.runProcessing_noThread()

        self.close()


    def runProcessing_noThread(self):
        start_time = time.time()
        if self.ZP_input: #This might be cutted down
            if (self.batRA and self.batDEC and self.batERROR
                and self.xrtRA and self.xrtDEC and self.xrtERROR):

                BAT = (self.batRA, self.batDEC, self.batERROR)
                XRT = (self.xrtRA, self.xrtDEC, self.xrtERROR)
                full_pipeline(self.path, BAT = BAT, XRT = XRT,
                                              ZP=self.ZP_input)

            elif self.batRA and self.batDEC and self.batERROR:
                BAT = (self.batRA, self.batDEC, self.batERROR)
                full_pipeline(self.path, BAT = BAT, ZP=self.ZP_input)

            else:
                XRT = (self.xrtRA, self.xrtDEC, self.xrtERROR)
                full_pipeline(self.path, XRT = XRT, ZP=self.ZP_input)

        else:
            if (self.batRA and self.batDEC and self.batERROR
                and self.xrtRA and self.xrtDEC and self.xrtERROR):

                BAT = (self.batRA, self.batDEC, self.batERROR)
                XRT = (self.xrtRA, self.xrtDEC, self.xrtERROR)
                full_pipeline(self.path, BAT = BAT, XRT = XRT)
            elif self.batRA and self.batDEC and self.batERROR:
                BAT = (self.batRA, self.batDEC, self.batERROR)
                full_pipeline(self.path, BAT = BAT)

            else:
                XRT = (self.xrtRA, self.xrtDEC, self.xrtERROR)
                full_pipeline(self.path, XRT = XRT)
        passed_time = int(round(time.time()-start_time, 0))


        #Sorting
        print("Sorting")

        start_time = time.time()
        SortPipeline(self.path)
        #plt.close("all")
        passed_time = int(round(time.time()-start_time, 0))

        #Pointsources
        print("Points")
        start_time = time.time()
        PointsAllCatsNFilters(self.path)
        #plt.close("all")
        passed_time = int(round(time.time()-start_time, 0))

        #Crossmacthing
        print("Crossmatching")
        start_time = time.time()
        crossmatch(self.path)
        passed_time = int(round(time.time()-start_time, 0))

        #Making infotable
        print("Table")
        start_time = time.time()
        ImportantInfoTable(self.path, "Gaia", "r")
        passed_time = int(round(time.time()-start_time, 0))
        print(passed_time)


        print("DONE dataprocessing")

        infowindow.setupUi(InfoWindow, self.path)
        InfoWindow.show()




#%%

    #Some part of the following is from StackOverflow by S.Nick.
    #Source for first part of the code:
    #https://stackoverflow.com/questions/50890645/ #continue in next line
    #resizing-table-widget-when-window-is-maximized
class Ui_InfoWindow(object):
    def setupUi(self, InfoWindow, path_in):
        InfoWindow.setObjectName("BasicInfo")
        self.centralwidget = QtWidgets.QWidget(InfoWindow)
        self.centralwidget.setObjectName("centralwidget")

        self.horizontalLayout_2 = QtWidgets.QHBoxLayout(self.centralwidget)
        self.horizontalLayout_2.setObjectName("horizontalLayout_2")
        self.verticalLayout_2 = QtWidgets.QVBoxLayout()
        self.verticalLayout_2.setObjectName("verticalLayout_2")
        self.horizontalLayout_2.addLayout(self.verticalLayout_2)

        self.tableWidget = QtWidgets.QTableWidget(self.centralwidget)
        self.tableWidget.setObjectName("tableWidget")
        self.verticalLayout_2.addWidget(self.tableWidget)
        self.tableWidget.setWindowTitle("Transactional Data")
        self.tableWidget.setColumnCount(7)
        self.tableWidget.setRowCount(5)
        self.tableWidget.setAlternatingRowColors(True)
        self.tableWidget.setHorizontalHeaderLabels(['A','B','C','D','E',
                                                                 'F','G'])
        self.tableWidget.horizontalHeader().setSectionResizeMode(
                                                QtWidgets.QHeaderView.Stretch)

        InfoWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(InfoWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 800, 24))
        self.menubar.setObjectName("menubar")
        InfoWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(InfoWindow)
        self.statusbar.setObjectName("statusbar")
        InfoWindow.setStatusBar(self.statusbar)

        self.moreinfoButton = QPushButton("More info")
        self.horizontalLayout_2.addWidget(self.moreinfoButton)


        self.retranslateUi(InfoWindow)
        QtCore.QMetaObject.connectSlotsByName(InfoWindow)


        #Variables
        self.path = path_in
        print(self.path)
        self.catalog = "Gaia"
        self.filter = "r"
        self.GRBName = self.path.split("/")[-2]
        self.pointsource = ""

        #Running functions
        self.LoadDataToTable()
        self.moreinfoButton.clicked.connect(self.moreInfo)


    def retranslateUi(self, InfoWindow):
        _translate = QtCore.QCoreApplication.translate
        InfoWindow.setWindowTitle(_translate("InfoWindow", "InfoWindow"))



    def LoadDataToTable(self):
        self.tableWidget.setRowCount(0) #Clearing table
        df = pd.read_csv(f"""{self.path}/FWHM/BasicInfo_{self.catalog}_
                                                    {self.filter}.csv""")
        self.tableWidget.setColumnCount(df.shape[1])
        self.tableWidget.setRowCount(df.shape[0])

        self.tableWidget.setHorizontalHeaderLabels(df.columns)
        for i in range(df.shape[0]):     #Row
            for j in range(df.shape[1]): #Column
                item = df.loc[i][j]
                if type(item) != str:
                    item = round(item, 4)
                item = QtWidgets.QTableWidgetItem(str(item))
                self.tableWidget.setItem(i, j, item)

                if 0 < j < 4 and item.text() == "Yes":
                     self.tableWidget.item(i, j).setBackground(QtGui.QColor(141
                                                                     , 232, 194))
                elif 4 < j and item.text() == "No":
                    self.tableWidget.item(i, j).setBackground(QtGui.QColor(141
                                                                     , 232, 194))
                else:
                    self.tableWidget.item(i, j).setBackground(QtGui.QColor(206
                                                                     , 87, 102))

    def moreInfo(self):
        print("clicked")
        #Setting up the window and giving the choosen path
        mainwindowUi.setupUi(MainWindow, self.path)
        MainWindow.show()


#%%

#Running the program
if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)

    FrontWindow = Ui_FrontWindow()
    FrontWindow.show()

    InfoWindow = QtWidgets.QMainWindow()
    infowindow = Ui_InfoWindow()

    MainWindow = QtWidgets.QMainWindow()
    mainwindowUi = Ui_MainWindow()


    sys.exit(app.exec_())
```

```python
import os

from pynot2.main import initialize #Step 1 from Pynot docu
from pynot2.phot_redux import run_pipeline #Step 2 from Pynot docu
from pynot2.transients import find_new_sources #Step 3 from Pynot docu


def arcminToDegress(arcmin):
    return arcmin/60

def arcsecToDegress(arcsec):
    return arcsec/3600

def getGRBfolders(path):
    GRBfolders = ([])
    for files in os.listdir(path):
        if "GRB" in files or "grb" in files:
            GRBfolders.append(files)
    return GRBfolders

def fixing_dataset(path, dataset_name = "dataset.pfc"):
    #Fixing dataset
    #Code snippet from
    """https://stackoverflow.com/questions/17140886/how-to-search-and-replace-
                                        text-in-a-file date: 12/12/2022"""
    f = open(dataset_name,'r')
    filedata = f.read()
    f.close()

    path_string = str(path)
    newdata = filedata.replace(fr"{path_string}", "")

    f = open(dataset_name,'w')
    f.write(newdata)
    f.close()


def find_new_sources_extended(path, BAT = "", XRT = "", search_catalog = "",
                                                                  ZP = ""):

    path_GRBs_subfolder = path + "imaging/"
    imagingGRBs = getGRBfolders(path_GRBs_subfolder)
    filters = ["r", "i", "g", "z"]
    for image in imagingGRBs:
        path_SDSS = path_GRBs_subfolder + image + "/"

        for file in os.listdir(path_SDSS):
            for filt in filters:
                if f"{image}_{filt}_SDSS.fits" == file:
                    old_path_SDSS_name = path_SDSS + f"""{image}_{filt}_
                                                 SDSS.fits"""
                    new_path_SDSS_name = path_SDSS + f"""{image}_{filt}_SDSS_
                                                 phot.fits"""

                    #This could maybe be a bit prettier?
                    #ZP provided
                    if ZP:
                        if BAT and XRT:
                            find_new_sources(old_path_SDSS_name,
                                             new_path_SDSS_name,
                                             search_catalog, # Gaia,2Mass,WISE
                                             loc_bat=BAT,
                                             loc_xrt=XRT,
                                             mag_lim=20.1,
                                             zp=ZP,
                                            )
                        elif BAT:
                            find_new_sources(old_path_SDSS_name,
                                             new_path_SDSS_name,
                                             search_catalog, # Gaia,2Mass,WISE
                                             loc_bat=BAT,
                                             mag_lim=20.1,
                                             zp=ZP,
                                            )
                        elif XRT:
                            find_new_sources(old_path_SDSS_name,
                                             new_path_SDSS_name,
                                             search_catalog, # Gaia,2Mass,WISE
                                             loc_xrt=XRT,
                                             mag_lim=20.1,
                                             zp=ZP,
                                            )
                        else:
                            print("ERROR; NO BAT OR XRT?!?!")

                    #No ZP provided
                    else:
                        if BAT and XRT:
                            find_new_sources(old_path_SDSS_name,
                                             new_path_SDSS_name,
                                             search_catalog, # Gaia,2Mass,WISE
                                             loc_bat=BAT,
                                             loc_xrt=XRT,
                                             mag_lim=20.1,
                                            )
                        elif BAT:
                            find_new_sources(old_path_SDSS_name,
                                             new_path_SDSS_name,
                                             search_catalog, # Gaia,2Mass,WISE
                                             loc_bat=BAT,
                                             mag_lim=20.1,
                                            )
                        elif XRT:
                            find_new_sources(old_path_SDSS_name,
                                             new_path_SDSS_name,
                                             search_catalog, # Gaia,2Mass,WISE
                                             loc_xrt=XRT,
                                             mag_lim=20.1,
                                            )
                        else:
                            print("ERROR; NO BAT OR XRT?!?!")


def full_pipeline(path, BAT="", XRT="", ZP=""):
    path = path + "/" #Just added this

    os.chdir(path)

    GRBfolders = getGRBfolders(path)
    for GRB in GRBfolders:
        GRBpath = path + str(GRB) + "/"
        initialize(GRBpath, "phot", pars_fname='parsGRB.yml')

        fixing_dataset(path)

        run_pipeline("parsGRB.yml")

        find_new_sources_extended(path, BAT, XRT, "Gaia", ZP)
        find_new_sources_extended(path, BAT, XRT, "2Mass", ZP)
        find_new_sources_extended(path, BAT, XRT, "WISE", ZP)

    os.chdir("..")
```

```python
import os
import shutil

def MakeSubFolders(path):
    new_folder = path + "/new_sources"
    if not(os.path.exists(new_folder)):
        os.mkdir(new_folder)
    Gaia_folder =  new_folder + "/Gaia"
    WISE_folder =  new_folder + "/WISE"
    Mass_folder =  new_folder + "/2Mass"
    if not(os.path.exists(Gaia_folder)):
        os.mkdir(Gaia_folder)
    if not(os.path.exists(WISE_folder)):
        os.mkdir(WISE_folder)
    if not(os.path.exists(Mass_folder)):
        os.mkdir(Mass_folder)

def SortNewSourceFiles(path):
    grbName = path.split("/")[-2]
    path_deep = path + f"imaging/{grbName}/"
    MakeSubFolders(path)

    #print(path_deep)
    catalogList = ["Gaia", "WISE", "2Mass"]
    png = "png"
    for file in os.listdir(path_deep):
        for cat in catalogList:
            to_get = f"{cat}_new_sources"
            if to_get in file and png in file:
                path_sending = path_deep + file
                path_receiving = path + f"new_sources/{cat}"
                shutil.copy2(path_sending, path_receiving)

def SortIntoFilters(path):
    cats = ["Gaia", "WISE", "2Mass"]
    path_new_sources = [path + f"new_sources/{cat}" for cat in cats]
    for subpath in path_new_sources:
        for file in os.listdir(subpath):
            cat = subpath.split("/")[-1]
            if cat in file:
                filt = file.split("_")[-2]
                new_folder = subpath + f"/{filt}"
                if not(os.path.exists(new_folder)):
                    os.mkdir(new_folder)
                if file not in os.listdir(new_folder):
                    path_file = subpath + f"/{file}"
                    shutil.move(path_file, new_folder)
                if file in os.listdir(subpath):
                    path_file = subpath + f"/{file}"
                    os.remove(path_file)


def SortPipeline(path):
    SortNewSourceFiles(path)
    SortIntoFilters(path)
```

```python
import os
import pandas as pd

def filtersForCrossmatching(path):
    filters = []
    for filename in os.listdir(path + "/FWHM"):
        filt_temp = filename.split("_")[-1][0]
        if filt_temp not in filters:
            filters.append(filt_temp)
    return filters


def crossmatch(path):
    filters = filtersForCrossmatching(path)
    for filt in filters:

        GRBName = path.split("/")[-2]
        dataframes = []
        ls_catalouge = ["Gaia", "WISE", "2Mass"]

        Ga, Wi, Ma = True, True, True
        for cat in ls_catalouge:
            path_temp = path + f"""/imaging/{GRBName}/{cat}_new_sources_
                                              {GRBName}_{filt}_SDSS.txt"""
            if os.path.exists(path_temp):
                df_temp = pd.read_fwf(path_temp, delimiter=" ")
                dataframes.append(df_temp)
                #print("Worked: ", cat, filt)
            else:
                NoDa = {'ra':["No data"],
                        'dec':["No data"]}
                NoDa_df = pd.DataFrame(NoDa)
                dataframes.append(NoDa_df)
                if cat == "Gaia":
                    Ga = False
                elif cat == "WISE":
                    Wi = False
                else:
                    Ma = False
                #print("Failed: ", cat, filt)

        Gaia_frame = dataframes[0]
        WISE_frame = dataframes[1]
        Mass_frame = dataframes[2]

        Gaia_ra, Gaia_dec = Gaia_frame["ra"], Gaia_frame["dec"]
        WISE_ra, WISE_dec = WISE_frame["ra"], WISE_frame["dec"]
        Mass_ra, Mass_dec = Mass_frame["ra"], Mass_frame["dec"]


        #Crossmatching lists
        #Comparing Gaia to WISE and 2Mass
        GaiaMatchesWISE = []
        GaiaMatches2Mass = []
        for i in range(1, len(Gaia_ra)):
            if "No data" in list(WISE_ra) or "No data" in list(WISE_dec):
                GaiaMatchesWISE.append("No data")
            elif (Gaia_ra[i] not in list(WISE_ra)
                  and Gaia_dec[i] not in list(WISE_dec)):
                GaiaMatchesWISE.append("Yes")
            else:
                GaiaMatchesWISE.append("No")

            if "No data" in list(Mass_ra) or "No data" in list(Mass_dec):
                GaiaMatches2Mass.append("No data")
            elif (Gaia_ra[i] not in list(Mass_ra)
                  and Gaia_dec[i] not in list(Mass_dec)):
                GaiaMatches2Mass.append("Yes")
            else:
                GaiaMatches2Mass.append("No")

        #Comparing WISE to Gaia and 2Mass
        WISEMatchesGaia = []
        WISEMatches2Mass = []
        for i in range(1, len(WISE_ra)):
            if "No Image" in list(Gaia_ra) or "No data" in list(Gaia_dec):
                WISEMatchesGaia.append("No data")
            elif (WISE_ra[i] not in list(Gaia_ra)
                  and WISE_dec[i] not in list(Gaia_dec)):
                WISEMatchesGaia.append("Yes")
            else:
                WISEMatchesGaia.append("No")

            if "No Image" in list(Mass_ra) or "No data" in list(Mass_dec):
                WISEMatches2Mass.append("No data")
            elif (WISE_ra[i] not in list(Mass_ra)
                  and WISE_dec[i] not in list(Mass_dec)):
                WISEMatches2Mass.append("Yes")
            else:
                WISEMatches2Mass.append("No")

        #Comparing 2Mass to Gaia and WISE
        MassMatchesGaia = []
        MassMatchesWISE = []
        for i in range(1, len(Mass_ra)):
            if "No Image" in list(Gaia_ra) or "No data" in list(Gaia_dec):
                MassMatchesGaia.append("No data")
            elif (Mass_ra[i] not in list(Gaia_ra)
                  and Mass_dec[i] not in list(Gaia_dec)):
                MassMatchesGaia.append("Yes")
            else:
                MassMatchesGaia.append("No")

            if "No Image" in list(WISE_ra) or "No data" in list(WISE_dec):
                MassMatchesWISE.append("No data")
            elif (Mass_ra[i] not in list(WISE_ra)
                  and Mass_dec[i] not in list(WISE_dec)):
                MassMatchesWISE.append("Yes")
            else:
                MassMatchesWISE.append("No")


        #Adding them to fwhm_dataframes.
        path_fwhm = path + "/FWHM"
        if Ga:
            dfGaia = pd.read_csv(path_fwhm + f"/fwhm_{GRBName}_Gaia_{filt}.csv"
                                                             , delimiter=",")
            dfGaia["In WISE"]  = GaiaMatchesWISE
            dfGaia["In 2Mass"] = GaiaMatches2Mass
            dfGaia.to_csv(path_fwhm + f"/fwhm_{GRBName}_Gaia_{filt}.csv",
                                                                  index=False)
```

```
if Wi:
    dfWISE = pd.read_csv(path_fwhm + f"/fwhm_{GRBName}_WISE_{filt}.csv"
                                                    , delimiter=",")
    dfWISE["In Gaia"]  = WISEMatchesGaia
    dfWISE["In 2Mass"] = WISEMatches2Mass
    dfWISE.to_csv(path_fwhm + f"/fwhm_{GRBName}_WISE_{filt}.csv",
                                                         index=False)

if Ma:
    df2Mass = pd.read_csv(path_fwhm + f"""/fwhm_{GRBName}_2Mass_
                                       {filt}.csv""", delimiter=",")
    df2Mass["In Gaia"] = MassMatchesGaia
    df2Mass["In WISE"] = MassMatchesWISE
    df2Mass.to_csv(path_fwhm + f"/fwhm_{GRBName}_2Mass_{filt}.csv",
                                                          index=False)
```

```python
import pandas as pd

def ImportantInfoTable(path, cat, filt):
    GRBName = path.split("/")[-2]
    path_SDSS = path + f"""/imaging/{GRBName}/{cat}_new_sources_{GRBName}_
                                                                    {filt}_SDSS.txt"""
    df_SDSS = pd.read_fwf(path_SDSS, delimiter=" ")

    path_fwhm = path + f"/FWHM/fwhm_{GRBName}_{cat}_{filt}.csv"
    df_fwhm = pd.read_csv(path_fwhm, delimiter=",")

    inBAT = []
    inXRT = []
    for i in df_SDSS["class"]:
        if i == 0:
            inBAT.append("No")
            inXRT.append("No")
        if i == 1:
            inBAT.append("Yes")
            inXRT.append("No")
        if i == 2:
            inBAT.append("Yes")
            inXRT.append("Yes")

    df_fwhm["BAT"]  = inBAT
    df_fwhm["XRT"]  = inXRT

    df_new = df_fwhm[["source", "BAT", "XRT", "Pointsource", "In WISE",
                                                                          "In 2Mass"]]

    df_new.to_csv(path + f"/FWHM//BasicInfo_{cat}_{filt}.csv", index=False)
```

```python
import os
from astropy.io import fits
import matplotlib.pyplot as plt
import numpy as np
from pynot2.transients import mad
from astropy.table import Table
import pandas as pd
from imexam.imexamine import Imexamine
import logging
import pickle
import shutil
import warnings
warnings.filterwarnings('ignore')


#%% Functions for FWHM calculations and pointsource selection
def calc_FWHM_gauss(sigmas):
    sig_x, sig_y = sigmas
    if sig_x==sig_y:
        return 2*np.sqrt(2*np.log(2)) * sig_x
    else:
        print("Sigmas are not the same")
        return None

def get_fwhm(image, xpix, ypix):
    plots=Imexamine()
    plots.set_data(image)
    plots.setlog(level=logging.CRITICAL)
    fwhm = []
    for i in range(len(xpix)):
        x, y = xpix[i], ypix[i]
        try:
            gauss_params = plots.line_fit(x, y, form='Gaussian1D',
                                          genplot = False)
            sigma = gauss_params[0].stddev
            fwhm_gauss  = calc_FWHM_gauss((sigma,sigma))
            fwhm.append(fwhm_gauss)
        except:
            pass

    return fwhm

def fwhm_range(fwhm_list):
    y, x, _ = plt.hist(fwhm_list, bins = 30)
    plt.close()

    tresh = 0.2
    fwhm = x[np.argmax(y)]
    fwhm_min = fwhm*(1-tresh)
    fwhm_max = fwhm*(1+tresh)
    return fwhm, fwhm_min, fwhm_max

def xypix_newsources(df_newsources, df_phot):
    df = df_newsources
    ra = np.array(df["ra"][1::]).astype(float)
    dec = np.array(df["dec"][1::]).astype(float)

    ra, ra_idx, ra_tab_idx    = np.intersect1d(np.round(ra, 5),
                                               np.round(np.array(df_phot["ra"]),
                                                        5), return_indices=True)
    dec, dec_idx, dec_tab_idx = np.intersect1d(np.round(dec, 5),
                                                np.round(np.array(df_phot["dec"]),
                                                         5), return_indices=True)
    tab_idx = np.intersect1d(ra_tab_idx, dec_tab_idx)

    x_newsources = df_phot["x"][tab_idx]
    y_newsources = df_phot["y"][tab_idx]
    return x_newsources, y_newsources

def pointsource_selection(fwhm_ls, image_path, newsources_path, df_phot_path):
    df_newsources = pd.read_fwf(newsources_path, delimiter=" ", )
    df_phot       = Table.read(df_phot_path)
    image, hdr    = fits.getdata(image_path, header = True)

    x_newsources, y_newsources = xypix_newsources(df_newsources, df_phot)

    fwhm_new_sources = get_fwhm(image, x_newsources, y_newsources)

    _, fwhm_min, fwhm_max = fwhm_range(fwhm_ls)

    pointsource_idx = list(np.where((fwhm_min < fwhm_new_sources) &
                                (fwhm_new_sources<fwhm_max))[0])

    return pointsource_idx, df_newsources


#%% Functions for open pickle data and recreate the transient image only with
    #pointsources

def openPickleDataFromPath(path):
    data = pickle.load(open(path, 'rb'))
    return data

"""This function is highly inspired by the transient.py files from PyNOT and
several things is directly copypasted and I do not take credit from it.
This functions purpose is only to recreate the image from the transient
detection just with pointsources only. The code is therefore very similar to
Jens Kroagers as huge parts is directly from his code to get the image to be
as close to his as possible. The code is kept here in appendix as the pipeline
wont work without this function, but credit should go to Jens Krogager"""
def recreatePlotWithOnlyPointsources(data_path, pointsources):
    data = openPickleDataFromPath(data_path)

    wcs = data["wcs"]
    img = data["img"]
    hdr = data["hdr"]
    search_catalog = data["search_catalog"]

    sattelite_data = data["sattelite_data"]
    sat_names = data["sat_names"]
    linestyles = data["linestyles"]
    colors = data["colors"]
    linewidths = data["linewidths"]

    #Most of the code underneath is directly from
    # transient.py with minor changes.
    fig = plt.figure(figsize=(9, 6))
    ax = fig.add_subplot(111, projection=wcs)
    med_val = np.nanmedian(img)
    ax.imshow(img, vmin=med_val-1*mad(img), vmax=med_val+10*mad(img),
              origin='lower', cmap=plt.cm.gray_r)
    ylims = ax.get_ylim()
    xlims = ax.get_xlim()

    ra_s, dec_s = data["marked_s"][0]

    ax.scatter(ra_s, dec_s, label=search_catalog,
              transform=ax.get_transform('fk5'),
              edgecolor='b', facecolor='none', s=40, marker='s')

    #This loop is directly taken from PyNOT
    for sat_name, burst_data, ls, col, lw in zip(sat_names, sattelite_data,
                                                  linestyles, colors,
                                                  linewidths):
        alpha, dec, sigma = burst_data
        err_circle = plt.Circle((alpha, dec), radius=sigma,
                                transform=ax.get_transform('fk5'),
                                facecolor='none', edgecolor=col, ls=ls, lw=lw,
                                label=sat_name)
        ax.add_patch(err_circle)

    if len(pointsources) > 0:
        #Does the transients marking.
        source_id = np.array(data["marked_trans"])[:,0]
        color     = np.array(data["marked_trans"])[:,1]
        mag       = np.array(data["marked_trans"])[:,2]
        band      = np.array(data["marked_trans"])[:,3]
        ra_t      = np.array(data["marked_trans"])[:,4]
        dec_t     = np.array(data["marked_trans"])[:,5]
        for k in range(len(source_id)):
            if int(source_id[k-1]) in pointsources:
                ax.scatter(float(ra_t[k]), float(dec_t[k]),
                           label='(%i)  %s = %.1f mag' % (int(source_id[k]),
                                                          band[k],
                                                          float(mag[k])),
                           transform=ax.get_transform('fk5'),
                           edgecolor=color[k],
                           facecolor='none', s=60, lw=1.5)
                ax.text(float(ra_t[k])-8/3600., float(dec_t[k])
                        , "%i" % (int(source_id[k])),
                        transform=ax.get_transform('fk5'))

    ax.set_xlabel("Right Ascension")
    ax.set_ylabel("Declination")
    box = ax.get_position()
    ax.set_position([0.15, box.y0, box.width, box.height])
    ax.legend(title=hdr['OBJECT'].upper(), loc='center left',
              bbox_to_anchor=(1, 0.5))
    ax.set_ylim(*ylims)
    ax.set_xlim(*xlims)

    return fig


#%% Functions for making folders, table, and saving the image

def make_fwhm_folder(initial_path):
    folder = initial_path + "/FWHM"
    if not(os.path.exists(folder)):
        os.mkdir(folder)

def make_fwhm_table(df_newsources, pointsource_idx):
    df = df_newsources
    cols_in  = ["source", "BAT", "XRT", "Pointsource"]
    data_in = {"source":      [],
               "BAT":          [],
               "XRT":          [],
               "Pointsource": []}

    for i in range(1, len(df)):
        data_in["source"].append(i)
        data_in["BAT"].append("Yes" if (df["class"][i]==1 or
                                        df["class"][i]==2) else "No")
        data_in["XRT"].append("Yes" if (df["class"][i]==2) else "No")
        data_in["Pointsource"].append("Yes" if i-1 in pointsource_idx else "No")

    df_temp = pd.DataFrame(data_in, columns = cols_in)
    df_temp.index +=1
    dataframe = pd.concat([df.drop(0), df_temp], axis=1)
    dataframe = dataframe[["source", "ra", "dec", "mag_auto", "BAT", "XRT",
                           "Pointsource"]]
    dataframe.index -=1
    return dataframe

def save_fwhm_table(initial_path, dataframe, GRBName, catalog, filt):
    folder = initial_path + "/FWHM"
    filename = folder + f'/fwhm_{GRBName}_{catalog}_{filt}.csv'
    dataframe.to_csv(filename, index=False)

def make_image_folder(initial_path):
    folder = initial_path + "/PointSources"
    if not(os.path.exists(folder)):
        os.mkdir(folder)

def make_save_pointsource_image(initial_path, pickledata_path, pointsource_idx,
                                GRBName, catalog, filt):
    #Spyder say fig is unused, but it is the figure which get saved.
    fig = recreatePlotWithOnlyPointsources(pickledata_path, pointsource_idx)
    folder = initial_path + "/PointSources"
    filename = f"PointSources_{GRBName}_{catalog}_{filt}.png"
    plt.savefig(os.path.join(folder, filename))
    plt.close()



#%% These functions sort the new_source_images into folder for catalog used
     #and subfolders for filters used

def MakeSubFolders(path):
    new_folder = path + "/new_sources"
    if not(os.path.exists(new_folder)):
        os.mkdir(new_folder)
    Gaia_folder =  new_folder + "/Gaia"
    WISE_folder =  new_folder + "/WISE"
    Mass_folder =  new_folder + "/2Mass"
    if not(os.path.exists(Gaia_folder)):
        os.mkdir(Gaia_folder)
    if not(os.path.exists(WISE_folder)):
        os.mkdir(WISE_folder)
    if not(os.path.exists(Mass_folder)):
        os.mkdir(Mass_folder)

def SortNewSourceFiles(path):
```

```python
    grbName = path.split("/")[-2]
    #print("Name: ", grbName)
    #print(path.split("/"))
    #print(path.split("/")[-2])
    #print()
    path_deep = path + f"imaging/{grbName}/"
    MakeSubFolders(path)

    #print(path_deep)
    catalogList = ["Gaia", "WISE", "2Mass"]
    png = "png"
    for file in os.listdir(path_deep):
        for cat in catalogList:
            to_get = f"{cat}_new_sources"
            if to_get in file and png in file:
                path_sending = path_deep + file
                path_receiving = path + f"new_sources/{cat}"
                shutil.copy2(path_sending, path_receiving)

def SortIntoFilters(path):
    cats = ["Gaia", "WISE", "2Mass"]
    path_new_sources = [path + f"new_sources/{cat}" for cat in cats]
    for subpath in path_new_sources:
        for file in os.listdir(subpath):
            cat = subpath.split("/")[-1]
            if cat in file:
                filt = file.split("_")[-2]
                new_folder = subpath + f"/{filt}"
                if not(os.path.exists(new_folder)):
                    os.mkdir(new_folder)
                if file not in os.listdir(new_folder):
                    path_file = subpath + f"/{file}"
                    shutil.move(path_file, new_folder)
                if file in os.listdir(subpath):
                    path_file = subpath + f"/{file}"
                    os.remove(path_file)


def SortPipeline(path):
    SortNewSourceFiles(path)
    SortIntoFilters(path)


#%% These functions make sure the fwhm pipeline is run in every catalog and
      #every filter.

#Get all the cats and filters based on the folder and subfolder in
  #path + /new_sources
def SearchFilterNFolders(path):
    newsources_path = path + "/new_sources"
    cats = os.listdir(newsources_path)
    filters = [os.listdir(newsources_path+f"/{cat}") for cat in cats]
    return tuple(zip(cats, filters))


def pointsource_pipe(initial_path, GRBName, catalog, filt, fwhm_ls):
    image_path      = initial_path + f"""imaging/{GRBName}/{GRBName}_
                                              {filt}_SDSS.fits"""
    df_phot_path    = initial_path + f"""imaging/{GRBName}/{GRBName}_
                                              {filt}_SDSS_phot.fits"""
    newsources_path = initial_path + f"""imaging/{GRBName}/{catalog}_
```

```
                                new_sources_{GRBName}_{filt}_SDSS.txt"""
    pickledata_path = initial_path + f"""imaging/{GRBName}/{catalog}_
                            new_sources_{GRBName}_{filt}_SDSS_data.pickle"""

    pointsource_idx, df_newsources = pointsource_selection(fwhm_ls, image_path,
                                                        newsources_path,
                                                        df_phot_path)
    dataframe = make_fwhm_table(df_newsources, pointsource_idx)

    make_fwhm_folder(initial_path)
    save_fwhm_table(initial_path, dataframe, GRBName, catalog, filt)

    make_image_folder(initial_path)
    make_save_pointsource_image(initial_path, pickledata_path, pointsource_idx,
                            GRBName, catalog, filt)


# Makes a fwhm_ls for each filter so it can be reused in each catalog as it is
# only when the filter changes the list does
# aswell, meaning it is slightly under 3 times faster.
def make_fwhm_ls_set(initial_path, catNfilters, GRBName):
    dic = {}
    for cat, filters in catNfilters:
        if len(filters) > 0:
            for filt in filters:
                if filt not in dic.keys():
                    image_path      = initial_path + f"""imaging/{GRBName}/
                                            {GRBName}_{filt}_SDSS.fits"""
                    df_phot_path    = initial_path + f"""imaging/{GRBName}/
                                        {GRBName}_{filt}_SDSS_phot.fits"""
                    newsources_path = initial_path + f"""imaging/{GRBName}/
                            {cat}_new_sources_{GRBName}_{filt}_SDSS.txt"""

                    df_newsources = pd.read_fwf(newsources_path,
                                            delimiter=" ", )
                    df_phot       = Table.read(df_phot_path)
                    image, hdr    = fits.getdata(image_path, header = True)

                    x_newsources, y_newsources = xypix_newsources(df_newsources
                                                            , df_phot)

                    fwhm_ls = get_fwhm(image, df_phot["x"], df_phot["y"])

                    dic[filt] = fwhm_ls
    return dic

#Loop through the catalogs and filters and calling them into the
   #pointsource finding function.
def PointsAllCatsNFilters(initial_path):
    GRBName = initial_path.split('/')[-2]
    catNfilters = SearchFilterNFolders(initial_path)

    dic = make_fwhm_ls_set(initial_path, catNfilters, GRBName)
    for cat, filters in catNfilters:
        if len(filters) > 0:
            for filt in filters:
                pointsource_pipe(initial_path, GRBName, cat, filt, dic[filt])
```

```python
from PyQt5 import QtCore, QtGui, QtWidgets
import os
import pandas as pd


class Ui_MainWindow(object):
    def setupUi(self, MainWindow, path_in):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(2920, 1440) #1440p

        #Catalogs
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.groupBox = QtWidgets.QGroupBox(self.centralwidget)
        self.groupBox.setGeometry(QtCore.QRect(70, 70, 131, 101))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.groupBox.setFont(font)
        self.groupBox.setObjectName("groupBox")
        #Gaia button
        self.radioButton = QtWidgets.QRadioButton(self.groupBox)
        self.radioButton.setGeometry(QtCore.QRect(20, 30, 101, 17))
        font = QtGui.QFont()
        font.setPointSize(10)
        self.radioButton.setFont(font)
        self.radioButton.setObjectName("radioButton")
        self.radioButton.setChecked(True)
        #WISE button
        self.radioButton_2 = QtWidgets.QRadioButton(self.groupBox)
        self.radioButton_2.setGeometry(QtCore.QRect(20, 50, 82, 17))
        font = QtGui.QFont()
        font.setPointSize(10)
        self.radioButton_2.setFont(font)
        self.radioButton_2.setObjectName("radioButton_2")
        #2Mass button
        self.radioButton_3 = QtWidgets.QRadioButton(self.groupBox)
        self.radioButton_3.setGeometry(QtCore.QRect(20, 70, 82, 17))
        font = QtGui.QFont()
        font.setPointSize(10)
        self.radioButton_3.setFont(font)
        self.radioButton_3.setObjectName("radioButton_3")

        #PointSourceSelection
        self.label = QtWidgets.QLabel(self.centralwidget)
        self.label.setGeometry(QtCore.QRect(70, 30, 311, 31))
        font = QtGui.QFont()
        font.setPointSize(14)
        font.setBold(True)
        font.setWeight(75)
        self.label.setFont(font)
        self.label.setObjectName("label")
        self.groupBox_2 = QtWidgets.QGroupBox(self.centralwidget)
        self.groupBox_2.setGeometry(QtCore.QRect(220, 70, 201, 101))
        font = QtGui.QFont()
        font.setPointSize(12)
        self.groupBox_2.setFont(font)
        self.groupBox_2.setObjectName("groupBox_2")
        #Yes button
        self.radioButton_4 = QtWidgets.QRadioButton(self.groupBox_2)
        self.radioButton_4.setGeometry(QtCore.QRect(20, 30, 82, 17))
```

```
font = QtGui.QFont()
font.setPointSize(10)
self.radioButton_4.setFont(font)
self.radioButton_4.setObjectName("radioButton_4")
#No button
self.radioButton_5 = QtWidgets.QRadioButton(self.groupBox_2)
self.radioButton_5.setGeometry(QtCore.QRect(20, 50, 82, 17))
font = QtGui.QFont()
font.setPointSize(10)
self.radioButton_5.setFont(font)
self.radioButton_5.setObjectName("radioButton_5")
self.radioButton_5.setChecked(True)

#Display Image
self.label_2 = QtWidgets.QLabel(self.centralwidget)
self.label_2.setGeometry(QtCore.QRect(70, 240, 1245, 831))
self.label_2.setText("")
self.label_2.setScaledContents(True)
self.label_2.setObjectName("label_2")

#List of filters
self.listWidget = QtWidgets.QListWidget(self.centralwidget)
self.listWidget.setGeometry(QtCore.QRect(510, 80, 201, 91))
self.listWidget.setObjectName("listWidget")
self.label_3 = QtWidgets.QLabel(self.centralwidget)
self.label_3.setGeometry(QtCore.QRect(450, 80, 47, 13))

#Table
self.tableWidget = QtWidgets.QTableWidget(self.centralwidget)
self.tableWidget.setGeometry(QtCore.QRect(1350, 240, 1025, 250))
self.tableWidget.setSelectionMode(
                QtWidgets.QAbstractItemView.ContiguousSelection)
self.tableWidget.setObjectName("tableWidget")
self.tableWidget.setColumnCount(2)
self.tableWidget.setRowCount(1)
self.tableWidget.horizontalHeader().setCascadingSectionResizes(False)
self.tableWidget.setEditTriggers(
                QtWidgets.QAbstractItemView.NoEditTriggers)



font = QtGui.QFont()
font.setPointSize(12)
self.label_3.setFont(font)
self.label_3.setObjectName("label_3")
MainWindow.setCentralWidget(self.centralwidget)
self.menubar = QtWidgets.QMenuBar(MainWindow)
self.menubar.setGeometry(QtCore.QRect(0, 0, 1768, 21))
self.menubar.setObjectName("menubar")
MainWindow.setMenuBar(self.menubar)
self.statusbar = QtWidgets.QStatusBar(MainWindow)
self.statusbar.setObjectName("statusbar")
MainWindow.setStatusBar(self.statusbar)

self.retranslateUi(MainWindow)
QtCore.QMetaObject.connectSlotsByName(MainWindow)


#Variables
self.path = path_in
```

```
        print(self.path)
        self.catalog = "Gaia"
        self.filter = "r"
        self.GRBName = self.path.split("/")[-2]
        self.pointsource = ""

        self.label.setText(f"GRB: {self.GRBName}")

        initpathname = f"""{self.path}/new_sources/{self.catalog}/
                          {self.filter}/{self.catalog}_new_sources_
                          {self.GRBName}_{self.filter}_SDSS.png"""
        self.label_2.setPixmap(QtGui.QPixmap(initpathname))
        self.listWidget.clear()
        self.listWidget.addItems(os.listdir(f"""{self.path}/new_sources/
                                                {self.catalog}"""))

        self.LoadDataToTable()

        #Functions
        self.radioButton.clicked.connect(self.Choose_GRBCat)
        self.radioButton_2.clicked.connect(self.Choose_GRBCat)
        self.radioButton_3.clicked.connect(self.Choose_GRBCat)
        self.radioButton_4.clicked.connect(self.ShowPointSourcesOnly)
        self.radioButton_5.clicked.connect(self.ShowPointSourcesOnly)

        self.listWidget.itemClicked.connect(self.ChooseFilter)


    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.groupBox.setTitle(_translate("MainWindow", "Catalouge"))
        self.radioButton.setText(_translate("MainWindow", "Gaia"))
        self.radioButton_2.setText(_translate("MainWindow", "WISE"))
        self.radioButton_3.setText(_translate("MainWindow", "2Mass"))
        #self.label.setText(_translate("MainWindow", "GRB: 2201thing"))
        self.groupBox_2.setTitle(_translate("MainWindow", """Show only
                                                        point sources"""))
        self.radioButton_4.setText(_translate("MainWindow", "Yes"))
        self.radioButton_5.setText(_translate("MainWindow", "No"))
        self.label_3.setText(_translate("MainWindow", "Filters: "))




    def LoadDataToTable(self):
        try:
            self.tableWidget.setRowCount(0) #Clearing table
            df = pd.read_csv(f"""{self.path}/FWHM/fwhm_{self.GRBName}_
                                {self.catalog}_{self.filter}.csv""")
            self.tableWidget.setColumnCount(df.shape[1])
            self.tableWidget.setRowCount(df.shape[0])
            self.tableWidget.setHorizontalHeaderLabels(df.columns)
            for i in range(df.shape[0]):     #Row
                for j in range(df.shape[1]): #Column
                    item = df.loc[i][j]
                    if type(item) != str:
                        item = round(item, 4)
                    item = QtWidgets.QTableWidgetItem(str(item))
                    self.tableWidget.setItem(i, j, item)
        except:
            pass

    def Choose_GRBCat(self):
        #Choose catalog
        if self.radioButton.isChecked():
            self.catalog = "Gaia"
        if self.radioButton_2.isChecked():
            self.catalog = "WISE"
        if self.radioButton_3.isChecked():
            self.catalog = "2Mass"


        if len(os.listdir(f"{self.path}/new_sources/{self.catalog}")) > 0:
            self.listWidget.clear()
            self.listWidget.addItems(os.listdir(f"""{self.path}/new_sources/
                                                 {self.catalog}"""))
        else:
            self.listWidget.clear()
            self.listWidget.addItem("Nothing found")

        if self.radioButton_5.isChecked():
            initpathname = f"""{self.path}/new_sources/{self.catalog}/
                                 {self.filter}/{self.catalog}_new_sources_
                                 {self.GRBName}_{self.filter}_SDSS.png"""
        elif self.radioButton_4.isChecked():
            initpathname = f"""{self.path}/PointSources/PointSources_
                             {self.GRBName}_{self.catalog}_{self.filter}.png"""

        self.label_2.setPixmap(QtGui.QPixmap(initpathname))

        self.LoadDataToTable()


    def ShowPointSourcesOnly(self):
        if self.radioButton_4.isChecked():
            print("yes is checked")
        if self.radioButton_5.isChecked():
            print("no is checked")

        print(self.filter)
        self.label_2.clear()
        if self.radioButton_5.isChecked():
            initpathname = f"""{self.path}/new_sources/{self.catalog}/
                                 {self.filter}/{self.catalog}_new_sources_
                                 {self.GRBName}_{self.filter}_SDSS.png"""
        elif self.radioButton_4.isChecked():
            initpathname = f"""{self.path}/PointSources/PointSources_
                             {self.GRBName}_{self.catalog}_{self.filter}.png"""

        self.label_2.setPixmap(QtGui.QPixmap(initpathname))


    def ChooseFilter(self, item):
        self.filter = item.text()
        self.label_2.clear()
        if self.radioButton_5.isChecked():
            initpathname = f"""{self.path}/new_sources/{self.catalog}/
                                 {self.filter}/{self.catalog}_new_sources_
                                 {self.GRBName}_{self.filter}_SDSS.png"""
        elif self.radioButton_4.isChecked():
            initpathname = f"""{self.path}/PointSources/PointSources_
```

```
                                {self.GRBName}_{self.catalog}_
                                {self.filter}.png"""

self.label_2.setPixmap(QtGui.QPixmap(initpathname))

self.LoadDataToTable()
```

```
# In[3]:

import os
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from tqdm import tqdm

from astroquery.ipac.irsa import Irsa
import astropy.coordinates as coord
import astropy.units as u
from PyAstronomy import pyasl

Irsa.ROW_LIMIT = 10000

# In[2]:
# # Defining functions

def random_point(min_ra, max_ra, min_dec, max_dec, size = 5):
    ra_random  = np.round(np.random.uniform(min_ra, max_ra, size),6)
    dec_random = np.round(np.random.uniform(min_dec, max_dec, size),6)
    ra_dec = np.array([ra_random, dec_random]).T
    return ra_dec

def is_sources_too_close(sep_cat, ref_cat, limit=1.5):
    matches = 0
    RA_ref, DEC_ref = np.array(ref_cat['ra']), np.array(ref_cat['dec'])
    RA_fake, DEC_fake = sep_cat[:,0], sep_cat[:,1]

    for i in range(len(RA_ref)):
        dist = pyasl.getAngDist(RA_fake, DEC_fake, RA_ref[i], DEC_ref[i])
        dist = dist[dist < limit/3600.]
        matches += len(dist)
    return matches

def arcminToDegress(arcmin):
    return arcmin/60

def arcsecToDegress(arcsec):
    return arcsec/3600

# In[3]:

def TestGalacticHeight(degrees, catalog, catalog_code, runs = 100, size = 5):
    catalog_folder = os.getcwd() + f"/{catalog}_catalog"
    degrees = float(degrees)
    try:
        tab = pd.read_csv(catalog_folder + f"/{catalog}_cat_gh{degrees}.csv",
                          delimiter=",")
    except:
        print("Downloading table")
        table = Irsa.query_region(coord.SkyCoord(0, degrees,
                                                  unit=(u.deg,u.deg),
                                                  frame='galactic'),
                                  catalog=catalog_code,
                                  spatial='Box',
                                  width= 4 * u.arcmin)
        #Fixing table
        r, d = list(table["ra"]), list(table["dec"])
        data = np.array([r,d]).T
        tab = pd.DataFrame(data, columns=['ra', 'dec'])

        #Saving the data
        if not(os.path.exists(catalog_folder)):
            os.mkdir(catalog_folder)
        name = f"{catalog}_cat_gh{degrees}"
        tab.to_csv(catalog_folder + f"/{name}.csv", index=False)



    #Finding the middle
    c = coord.SkyCoord(0, degrees,
                       unit=(u.deg,u.deg),
                       frame='galactic').icrs
    ra_midt = c.ra.deg
    dec_midt = c.dec.deg

    #Finding the bounderies for random points
    min_ra  = ra_midt - arcminToDegress(2) * 1/np.cos(dec_midt *np.pi/180)
    max_ra  = ra_midt + arcminToDegress(2) * 1/np.cos(dec_midt *np.pi/180)
    min_dec = dec_midt - arcminToDegress(2)
    max_dec = dec_midt + arcminToDegress(2)

    n_s = []
    for i in range(runs):
        random_particles = random_point(min_ra, max_ra, min_dec, max_dec,
                                        size = size) #Creating random points
        n_s.append(is_sources_too_close(random_particles, tab,
                                        limit=1.5)) #Finding number of matches

    n = np.mean(n_s) #Taking the mean of the number of matches
    error = np.std(n_s) #Taking the standard deviation of the number of matches

    return n, error, len(tab), tab, random_particles

#Testing the code for one angle.
n, error, cat_length, cat_table, random_particles = TestGalacticHeight(
                                                    degrees = 15,
                                                    catalog = "gaia",
                                                    catalog_code = 'gaia_edr3_source',
                                                    runs = 100,
                                                    size = 1000)
print("n_mathces: ", n, "Errors: ", error, "Lenght of Gaia: ", cat_length)

# In[4]:

#Function to run it with different angles
def bigrun(gh_angels, catalog, catalog_code, runs, size_of_runs):
    n_gh = []
    error_gh = []
    l_gh = []
    for gh_deg in tqdm(gh_angels):
        n_matches, error, table_length, tab, random_particles = TestGalacticHeight(degre
                                                                         catalog = catalog
                                                                         catalog_code = ca
                                                                         runs = runs,
                                                                         size = size_of_ru
        """
        This is the part that cannot be seen.
        TestGalacticHeight(degrees = gh_deg,
                           catalog = catalog,
                            catalog_code = catalog_code,
                            runs = runs,
                            size = size_of_runs)
        """
        n_gh.append(n_matches)
        error_gh.append(error)
        l_gh.append(table_length)
    return n_gh, error_gh, l_gh

# In[5]:

# # Running simulations
# Galactic heigt angles
gh = np.arange(-89, 89.9, 1)

# In[14]:

#Running for Gaia
n_gh_gaia, error_gh_gaia, l_gh_gaia = bigrun(gh, "gaia", 'gaia_edr3_source',
                                             50, 20000)

# In[15]:

#Saving data such it doesent have to be run again.
gaia_data = {"gh": gh,
             "n_gh_gaia": n_gh_gaia,
             "error_gh_gaia": error_gh_gaia,
             "l_gh_gaia": l_gh_gaia}
df = pd.DataFrame.from_dict(gaia_data)
df.to_csv(os.getcwd() + 'gaia_data_gh_test.csv')

# In[16]:

#Running for WISE
n_gh_wise, error_gh_wise, l_gh_wise = bigrun(gh, "WISE", 'allwise_p3as_psd',
                                             50, 20000)

# In[17]:

#Saving data such it doesent have to be run again.
wise_data = {"gh": gh,
             "n_gh_wise": n_gh_wise,
             "error_gh_wise": error_gh_wise,
             "l_gh_wise": l_gh_wise}
df = pd.DataFrame.from_dict(wise_data)
df.to_csv(os.getcwd() + 'wise_data_gh_test.csv')

# In[18]:

#Running for 2MASS
n_gh_2mass, error_gh_2mass, l_gh_2mass = bigrun(gh, "2Mass", 'fp_psc',
                                                50, 20000)

# In[19]:

#Saving data such it doesent have to be run again.
mass_data = {"gh": gh,
             "n_gh_2mass": n_gh_2mass,
             "error_gh_2mass": error_gh_2mass,
             "l_gh_2mass": l_gh_2mass}
df = pd.DataFrame.from_dict(mass_data)
```

```python
df.to_csv(os.getcwd() + '2mass_data_gh_test.csv')

# In[18]:

# # Plotting data and saving it
#Functions that does plotting
def plotting_data(gh_a, n_gh, error, l_gh, catalog, size_of_runs):
    SIZE = size_of_runs/100
    cat_name = catalog.capitalize() if (catalog[0].isdigit() == False
                                and catalog[0].isupper() == False) else catalog

    title_fontsize = 14
    label_fontsize = 12


    plt.figure(figsize = (10,3))
    plt.errorbar(gh, np.array(n_gh)/SIZE, yerr = np.array(error)/SIZE,
                 fmt=" ", label = "Errorbars", color = "red")
    plt.plot(gh, np.array(n_gh)/SIZE, '.',
             label = "Data", color = "C0")
    plt.title(f"""Number of matches as function of galactic height with
              {cat_name} \n Used box-search:
                  width = 4 arcmin""", fontsize=title_fontsize)
    plt.xlabel("Galactic height [degress]", fontsize = label_fontsize)
    plt.ylabel("Number of matches in percent", fontsize = label_fontsize)
    #plt.ylim(0, 1.8)
    plt.legend()
    plt.savefig(f"Figs/{cat_name}_gh_Matches.jpg", dpi = 600,
                bbox_inches='tight')

    plt.figure(figsize = (10,3))
    plt.plot(gh, l_gh, '.')
    plt.title(f"""Number of objects in {cat_name} as function of galactic
              height \n Used box-search: width = 4 arcmin""",
              fontsize=title_fontsize)
    plt.xlabel("Galactic height [degress]", fontsize = label_fontsize)
    plt.ylabel(r"Objects in catalog per 16/cos($dec_{icrs}$) ${arcmin^2}$",
               fontsize = label_fontsize)
    plt.savefig(f"Figs/{cat_name}_gh_Objects.jpg", dpi = 600,
                bbox_inches='tight')

# In[19]:

#Open the data and plotting it
df = pd.read_csv(os.getcwd() + 'gaia_data_gh_test.csv')
gh, n_gh_gaia = df["gh"], df["n_gh_gaia"]
error_gh_gaia, l_gh_gaia = df["error_gh_gaia"], df["l_gh_gaia"]
plotting_data(gh, n_gh_gaia, error_gh_gaia, l_gh_gaia, "gaia", 20000)

# In[20]:

#Open the data and plotting it
df = pd.read_csv(os.getcwd() + 'wise_data_gh_test.csv')
gh, n_gh_wise = df["gh"], df["n_gh_wise"]
error_gh_wise, l_gh_wise = df["error_gh_wise"], df["l_gh_wise"]
plotting_data(gh, n_gh_wise, error_gh_wise, l_gh_wise, "WISE", 20000)

# In[21]:

#Open the data and plotting it
df = pd.read_csv(os.getcwd() + '2mass_data_gh_test.csv')
```

```
gh, n_gh_2mass  = df["gh"], df["n_gh_2mass"]
error_gh_2mass, l_gh_2mass = df["error_gh_2mass"], df["l_gh_2mass"]
plotting_data(gh, n_gh_2mass, error_gh_2mass, l_gh_2mass, "2MASS", 20000)

# In[4]:

# # Plotting the data with zoom on the y-axis
def plotting_data_zoomed(gh_a, n_gh, error, l_gh, catalog, size_of_runs):
    SIZE = size_of_runs/100
    cat_name = catalog.capitalize() if (catalog[0].isdigit() == False
                                        and catalog[0].isupper() == False) else catalog

    title_fontsize = 14
    label_fontsize = 12


    plt.figure(figsize = (10,3))
    plt.errorbar(gh, np.array(n_gh)/SIZE, yerr = np.array(error)/SIZE,
                 fmt=" ", label = "Errorbars", color = "red")
    plt.plot(    gh, np.array(n_gh)/SIZE, '.', label = "Data",
             color = "C0")
    plt.title(f"""Number of matches as function of galactic height with
              {cat_name} \n Used box-search: width = 4 arcmin""",
              fontsize=title_fontsize)
    plt.xlabel("Galactic height [degress]", fontsize = label_fontsize)
    plt.ylabel("Number of matches in percent \n Box width = 4 arcmin",
               fontsize = label_fontsize)
    plt.ylim(0, 1.8)
    plt.legend()
    plt.savefig(f"Figs/{cat_name}_gh_Matches_zoomed.jpg", dpi = 600,
                bbox_inches='tight')

    plt.figure(figsize = (10,3))
    plt.plot(gh, l_gh, '.')
    plt.title(f"Number of objects in {cat_name} as function of galactic height"
              , fontsize=title_fontsize)
    plt.xlabel("Galactic height [degress]", fontsize = label_fontsize)
    plt.ylabel(r"Objects in catalog per 16/cos($dec_{icrs}$) ${arcmin^2}$",
               fontsize = label_fontsize)
    plt.ylim(0, 140)
    plt.savefig(f"Figs/{cat_name}_gh_Objects_zoomed.jpg",
                dpi = 600, bbox_inches='tight')

# In[5]:

#Open the data and plotting it
df = pd.read_csv(os.getcwd() + 'gaia_data_gh_test.csv')
gh, n_gh_gaia = df["gh"], df["n_gh_gaia"]
error_gh_gaia, l_gh_gaia = df["error_gh_gaia"], df["l_gh_gaia"]
plotting_data_zoomed(gh, n_gh_gaia, error_gh_gaia, l_gh_gaia, "gaia", 20000)


# In[6]:

#Open the data and plotting it
df = pd.read_csv(os.getcwd() + '2mass_data_gh_test.csv')
gh, n_gh_2mass, error_gh_2mass, l_gh_2mass = df["gh"], df["n_gh_2mass"]
error_gh_2mass, l_gh_2mass = df["error_gh_2mass"], df["l_gh_2mass"]
plotting_data_zoomed(gh, n_gh_2mass, error_gh_2mass, l_gh_2mass,
                     "2MASS", 20000)
```

```
# In[1]:


import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from pynot.transients import mad
from tqdm import tqdm
from PyAstronomy import pyasl

from astropy.io import fits
from astropy.table import Table
from astropy.coordinates import SkyCoord
from astropy.nddata import Cutout2D
from astropy.modeling.fitting import LevMarLSQFitter
from astropy.stats import sigma_clipped_stats
from astropy.nddata import NDData
from astropy.visualization import simple_norm

from photutils.psf import EPSFBuilder, extract_stars
from photutils.psf import IterativelySubtractedPSFPhotometry
from photutils.background import MMMBackground
from photutils.detection import IRAFStarFinder
from photutils.psf import DAOGroup, IntegratedGaussianPRF

import pickle
import warnings
warnings.filterwarnings('ignore')


def arcminToDegress(arcmin):
    return arcmin/60

def arcsecToDegress(arcsec):
    return arcsec/3600


# In[2]:

path = os.getcwd()

path_pickle_1 = path + """\GRB220521A_1\imaging\grb220521/Gaia_new_sources_
                                                 grb220521_r_SDSS_data.pickle"""
path_temp_1 = """/GRB220521A_1/imaging/grb220521/Gaia_new_sources_grb220521_r_
                                                                     SDSS.txt"""
raw_dataframe_1 = pd.read_fwf(path + path_temp_1)
phot_path_1 = path + """\GRB220521A_1\imaging\grb220521/grb220521_r_SDSS_
                                                                 phot.fits"""
phot_1 = Table.read(phot_path_1)

path_pickle_2 = path + """\GRB220521A_2\imaging\grb220521/Gaia_new_sources_
                                                 grb220521_r_SDSS_data.pickle"""
path_temp_2   = """/GRB220521A_2/imaging/grb220521/Gaia_new_sources_grb220521
                                                                 _r_SDSS.txt"""
raw_dataframe_2   = pd.read_fwf(path + path_temp_2)
phot_path_2 = path + """\GRB220521A_2\imaging\grb220521/grb220521_r_
                                                             SDSS_phot.fits"""
phot_2 = Table.read(phot_path_2)
```

```python
path_pickle_3 = path + """\GRB220521A_3\imaging\grb220521/Gaia_new_sources_
                                        grb220521_r_SDSS_data.pickle"""
path_temp_3   = """/GRB220521A_3/imaging/grb220521/Gaia_new_sources_
                                                grb220521_r_SDSS.txt"""
raw_dataframe_3   = pd.read_fwf(path + path_temp_3)
phot_path_3 = path + """\GRB220521A_3\imaging\grb220521/grb220521_r_SDSS
                                                            _phot.fits"""
phot_3 = Table.read(phot_path_3)

image_path1 = path + "\GRB220521A_1\imaging\grb220521/grb220521_r_SDSS.fits"
image1, hdr1 = fits.getdata(image_path1, header = True)

image_path2 = path + "\GRB220521A_1\imaging\grb220521/grb220521_r_SDSS.fits"
image2, hdr2 = fits.getdata(image_path2, header = True)

image_path3 = path + "\GRB220521A_1\imaging\grb220521/grb220521_r_SDSS.fits"
image3, hdr3 = fits.getdata(image_path3, header = True)


print(len(raw_dataframe_1), len(raw_dataframe_2), len(raw_dataframe_3))

round_nr = 4 #Rounding for crossmatching
def reducingcoordinates(data, round_nr):
    ls = []
    for i in tqdm(range(1, len(data))):
        ra, dec = data[["ra", "dec"]].iloc[i]
        ra, dec = round(float(ra), round_nr), round(float(dec),round_nr)
        ls.append([ra, dec])
    return ls

dataframe_1 = reducingcoordinates(raw_dataframe_1, round_nr)
dataframe_2 = reducingcoordinates(raw_dataframe_2, round_nr)
dataframe_3 = reducingcoordinates(raw_dataframe_3, round_nr)


# In[3]:

#Crossmatch sources between images
def crossmatching(data, data_refs):
    n_data_refs = len(data_refs)

    coordinates = []
    for radec in data:
        temp_counter = 0
        for j in range(n_data_refs):
            if radec not in data_refs[j]:
                break
            else:
                temp_counter += 1

            if temp_counter == n_data_refs:
                coordinates.append(radec)

    return coordinates

coordinates_1 = crossmatching(dataframe_1, [dataframe_2, dataframe_3])
coordinates_2 = crossmatching(dataframe_2, [dataframe_1, dataframe_3])
coordinates_3 = crossmatching(dataframe_3, [dataframe_1, dataframe_2])


# In[5]:
```

```python
#Getting indexes from the raw frames of the sources
def raw_frame_index(coordinates, raw_frame, round_nr):
    idx = []
    rd  = []
    for i in range(1, len(raw_frame)):
        ra, dec = raw_frame[["ra", "dec"]].iloc[i]
        radec = [round(float(ra), round_nr), round(float(dec),round_nr)]

        if radec in coordinates:
            idx.append(i)
            rd.append(radec)

    return idx, rd

idx_1, rd_1 = raw_frame_index(coordinates_1, raw_dataframe_1, round_nr)
idx_2, rd_2 = raw_frame_index(coordinates_2, raw_dataframe_2, round_nr)
idx_3, rd_3 = raw_frame_index(coordinates_3, raw_dataframe_3, round_nr)

print(len(idx_1), len(idx_2), len(idx_3))

# In[6]:


def openPickleDataFromPath(path):
    data = pickle.load(open(path, 'rb'))
    return data

#Most of this functions should be creditted to Jens Kroager as it is from PyNOT
def create_plot(pickle_path, raw_frame, index, idx_grb):
    data = openPickleDataFromPath(pickle_path)

    wcs = data["wcs"]
    img = data["img"]
    hdr = data["hdr"]

    sattelite_data = data["sattelite_data"]
    sat_names = data["sat_names"]
    linestyles = data["linestyles"]
    colors = data["colors"]
    linewidths = data["linewidths"]

    #Most of the code underneath is directly from
    #   transient.py with minor changes.
    fig = plt.figure(figsize=(9, 6))
    ax = fig.add_subplot(111, projection=wcs)
    med_val = np.nanmedian(img)
    ax.imshow(img, vmin=med_val-1*mad(img), vmax=med_val+10*mad(img),
              origin='lower', cmap=plt.cm.gray_r)
    ylims = ax.get_ylim()
    xlims = ax.get_xlim()


    #This loop is directly taken from PyNOT
    for sat_name, burst_data, ls, col, lw in zip(sat_names, sattelite_data,
                                                  linestyles, colors, linewidths):
        alpha, dec, sigma = burst_data
        err_circle = plt.Circle((alpha, dec), radius=sigma,
                                transform=ax.get_transform('fk5'),
                                facecolor='none', edgecolor=col, ls=ls, lw=lw,
                                label=sat_name)
        ax.add_patch(err_circle)


    source_id = np.array(data["marked_trans"])[:,0]
    color     = np.array(data["marked_trans"])[:,1]
    mag       = np.array(data["marked_trans"])[:,2]
    band      = np.array(data["marked_trans"])[:,3]
    ra_t      = np.array(data["marked_trans"])[:,4]
    dec_t     = np.array(data["marked_trans"])[:,5]

    #This is also taken from PyNOT
    for k in range(len(source_id)):
        if int(source_id[k]) in index or int(source_id[k]) == idx_grb:
            ax.scatter(float(ra_t[k]), float(dec_t[k]),
                       label='(%i)  %s = %.1f mag' % (int(source_id[k]),
                                                       band[k], float(mag[k])),
                       transform=ax.get_transform('fk5'), edgecolor=color[k],
                       facecolor='none', s=60, lw=1.5)
            ax.text(float(ra_t[k])-8/3600., float(dec_t[k]), "%i" %
                    (int(source_id[k])),
                    transform=ax.get_transform('fk5'))

    ax.set_xlabel("Right Ascension")
    ax.set_ylabel("Declination")
    box = ax.get_position()
    ax.set_position([0.15, box.y0, box.width, box.height])
    ax.legend(title=hdr['OBJECT'].upper(), loc='center left',
              bbox_to_anchor=(1, 0.5))
    ax.set_ylim(*ylims)
    ax.set_xlim(*xlims)


create_plot(path_pickle_1, raw_dataframe_1, idx_1, 474)
create_plot(path_pickle_2, raw_dataframe_2, idx_2, 538)
create_plot(path_pickle_3, raw_dataframe_3, idx_3, 325)


# In[7]:

def radec_coords(raw_frame, index, idx_grb):
    ra, dec = [], []

    for idx in index:
        ra_t, dec_t = (float(raw_frame.iloc[idx]["ra"]),
                        float(raw_frame.iloc[idx]["dec"]))
        ra.append(ra_t)
        dec.append(dec_t)

    ra_grb, dec_grb = (float(raw_frame.iloc[idx_grb]["ra"]),
                        float(raw_frame.iloc[idx_grb]["dec"]))
    ra.append(ra_grb)
    dec.append(dec_grb)

    return ra, dec

radec1 = radec_coords(raw_dataframe_1, idx_1, 474)
radec2 = radec_coords(raw_dataframe_2, idx_2, 538)
radec3 = radec_coords(raw_dataframe_3, idx_3, 325)


# In[8]:
```

```
#Getting the xy pixels
def pix_coords(radec, phot):
    x, y = [], []
    radec = np.array(radec).T
    for ra, dec in radec:
        i = np.where((np.round(phot["ra"], 5) == ra) & (np.round(phot["dec"],
                                                                 5) == dec))[0][0]
        x.append(phot[i]["x"])
        y.append(phot[i]["y"])
    xy = np.array([x,y])
    return xy

xy1 = pix_coords(radec1, phot_1)
xy2 = pix_coords(radec2, phot_2)
xy3 = pix_coords(radec3, phot_3)
len(xy1[0]), len(xy2[0]), len(xy3[0])

# In[9]:

#Function to cutout small part of the image, like a star.
def cutout_func_stars(image, position, xy_size):
    x_size, y_size = xy_size
    cutout = Cutout2D(image, position, (x_size, y_size))
    reduced_image = cutout.data
    return reduced_image

#Function to get the PSF of the sources
def get_psf_flux_sources(radec, xy, image, header, photometry):
    ra, dec  = [], []
    x_0      = []
    y_0      = []
    flux_psf = []
    flux_error = []

    x, y = xy
    ras, decs = radec

    size = 25
    for i in tqdm(range(len(x))):
        selected_star = cutout_func_stars(image, position = (x[i], y[i]),
                                          xy_size = (size, size))
        result_tab = photometry(image=selected_star)


        flux_unc = sum(result_tab["flux_unc"])
        flux_error.append(flux_unc)
        flux = sum(result_tab["flux_fit"])
        flux_psf.append(flux)
        x_0.append(x[i])
        y_0.append(y[i])
        ra.append(ras[i])
        dec.append(decs[i])


    result_dic = {}
    result_dic["ra"] = ra
    result_dic["dec"] = dec
    result_dic["x_0"] = x_0
    result_dic["y_0"] = y_0
    result_dic["flux_psf"] = flux_psf
    result_dic["flux_error"] = flux_error

    result_table = Table(result_dic)

    return result_table


# In[10]:


path = os.getcwd()

image_path1 = path + "\GRB220521A_1\imaging\grb220521/grb220521_r_SDSS.fits"
image1, hdr1 = fits.getdata(image_path1, header = True)

image_path2 = path + "\GRB220521A_2\imaging\grb220521/grb220521_r_SDSS.fits"
image2, hdr2 = fits.getdata(image_path2, header = True)

image_path3 = path + "\GRB220521A_3\imaging\grb220521/grb220521_r_SDSS.fits"
image3, hdr3 = fits.getdata(image_path3, header = True)


def get_grb_mag_psf(radec, xy, re_image, header):

    mean_val, median_val, std_val = sigma_clipped_stats(re_image, sigma=2.0)
    re_image -= median_val
    size = 25
    hsize = (size - 1) / 2


    #Making the PSF function which then can be applied on the sources
    #The follow 34 lines are almost direcly from the photulis documentation:
    #    https://photutils.readthedocs.io/en/stable/psf.html
    x, y = xy
    mask = ((x > hsize) & (x < (re_image.shape[1] -1 - hsize)) &
            (y > hsize) & (y < (re_image.shape[0] -1 - hsize)))

    stars_tbl = Table()
    stars_tbl['x'] = x[mask]
    stars_tbl['y'] = y[mask]

    data = NDData(data=re_image)
    stars = extract_stars(data, stars_tbl, size=size)
    print(len(stars))

    epsf_builder = EPSFBuilder(oversampling=4,
                               maxiters=10,
                               smoothing_kernel = "quadratic",
                               progress_bar=True)
    epsf, fitted_stars = epsf_builder(stars)

    daogroup = DAOGroup(crit_separation=8)
    background = MMMBackground()
    iraffind = IRAFStarFinder(threshold=2.5 *background(re_image), fwhm=4.5)
    fitter = LevMarLSQFitter()
    gaussian_prf = IntegratedGaussianPRF(sigma=2.05)
    gaussian_prf.sigma.fixed = False

    photometry = IterativelySubtractedPSFPhotometry(finder=iraffind,
                                                    group_maker=daogroup,
                                                    bkg_estimator=background,
                                                    psf_model=epsf,
                                                    fitter=fitter,
                                                    fitshape=(11, 11),
                                                    niters=2)
    epsf.x_0.fixed = True
    epsf.y_0.fixed = True
   #The above are from the https://photutils.readthedocs.io/en/stable/psf.html

    flux_psf_results = get_psf_flux_sources(radec, xy, re_image, header,
                                            photometry)

    position = SkyCoord('18h20m55.12s +10d22m20.6s', frame='icrs')
    selected_star = cutout_func_stars(re_image, position = (745.1694651022076,
                                                            739.9022987169562)
                                      , xy_size = (size, size))

    norm = simple_norm(selected_star, 'log', percent=99.0)
    plt.imshow(selected_star, norm=norm, origin='lower', cmap='viridis')


    magnitudes = 27.5 - 2.5*np.log10(abs(flux_psf_results["flux_psf"]))
    flux_psf_results["mag_psf"] = magnitudes

    grb_indexes = []
    for i in range(len(flux_psf_results)):
        if pyasl.getAngDist(flux_psf_results["ra"][i],
                            flux_psf_results["dec"][i], 275.2297,
                            10.3724) < arcsecToDegress(2.5):
            grb_indexes.append(i)

    grb_mag = flux_psf_results[grb_indexes]["mag_psf"]

    return grb_mag, flux_psf_results, grb_indexes, mask


grb_mag1, table1, grb_idx1, mask1 = get_grb_mag_psf(radec1, xy1, image1, hdr1)
grb_mag2, table2, grb_idx2, mask2 = get_grb_mag_psf(radec2, xy2, image2, hdr2)
grb_mag3, table3, grb_idx3, mask3 = get_grb_mag_psf(radec3, xy3, image3, hdr3)

table1 = table1[mask3]
table2 = table2[mask3]
table3 = table3[mask3]


# In[24]:

#Plotting the spread of the magnitudes.
def mags_plot(grb_idx, dfs):
    mags = []
    for df in dfs:
        mags.append(df["mag_psf"])
    mags = np.array(mags)

    image_n_list = list(range(1, len(dfs)+1))
    plt.figure(figsize =(6,6))
    for i in range(len(mags[0,:])):
        mags_temp = np.array(mags[:,i]) - np.median(mags[:,i])
        if i != grb_idx:
            plt.plot(image_n_list, mags_temp, "-", color = "grey")

    plt.plot(image_n_list, mags_temp, "-", color = "grey",
             label = "Reference sources") #This is just to get a label
    mags_temp = np.array(mags[:,grb_idx]) - np.median(mags[:,grb_idx])
    plt.plot(image_n_list, mags_temp, "-", color = "green",
             label = "Transient")
    plt.xlabel("Image N")
    plt.ylabel("Magnitude")
    plt.legend()
    plt.xticks(image_n_list)
    plt.savefig("Spread_and_PSF.png", dpi = 400)

    mags_mean = np.array([np.mean(df["mag_psf"]) for df in dfs])

    diffs = mags_mean - np.array(mags[:,grb_idx])
    print("Mean of magnitudes: ", mags_mean)
    print()
    print("grb magnitude in each image: ", mags[:,grb_idx])
    print("grb magnitude diffs:         ",
          mags[:,grb_idx][0]-mags[:,grb_idx][1],
          mags[:,grb_idx][1]-mags[:,grb_idx][2])
    print()
    print("grb diff from mean: ", diffs)
    print("grb drop:       ", diffs[1]-diffs[0], diffs[2]-diffs[1])

    return mags

#Getting index of the GRB
def get_grb_index(new_sources, xrt_pos = (275.2297, 10.3724), max_dist = 2.5):
    ra, dec = new_sources["ra"], new_sources["dec"]
    xrt_ra, xrt_dec = xrt_pos
    grb = new_sources[pyasl.getAngDist(ra, dec, xrt_ra,
                                       xrt_dec) < arcsecToDegress(max_dist)]
    if len(grb) == 1:
        grb_index = np.where(new_sources[["ra", "dec"]] == grb[["ra", "dec"]])
        return grb_index[0][0]
    else:
        print(f"ERROR: Too many or few in proximity. Found: {len(grb)}")

print(get_grb_index(table1))
grb_index = 25

new_sources = [table1, table2, table3]
mags = mags_plot(grb_index, new_sources)


# In[13]:

#Getting the fluxes and calculates the magnitudes as well as the error
def get_flux_and_fluxerror(new_sources):
    flux       = new_sources["flux_psf"]
    flux_error = new_sources["flux_error"]
    return flux, flux_error


def mag_err_calc(flux, flux_err):
    mag_err = 2.5/np.log(10) * flux_err/flux
    return mag_err


def mag_err_calc_all(flux_list, flux_error):
    flux  = np.array(flux_list)
    error = np.array(flux_error)
    mag_error = mag_err_calc(flux, error)
    return mag_error

def combine_mag_error_calcs(sources, idxs):
    flux, flux_error = get_flux_and_fluxerror(sources[idxs])
    return mag_err_calc_all(flux, flux_error)

ref_idxs = []
for i in range(len(table1)):
    if i != grb_index:
        ref_idxs.append(i)

mag_err_trans1 = combine_mag_error_calcs(table1, [grb_index])
mag_err_refs1  = combine_mag_error_calcs(table1, ref_idxs)

mag_err_trans2 = combine_mag_error_calcs(table2, [grb_index])
mag_err_refs2  = combine_mag_error_calcs(table2, ref_idxs)

mag_err_trans3 = combine_mag_error_calcs(table3, [grb_index])
mag_err_refs3  = combine_mag_error_calcs(table3, ref_idxs)

mag_err_trans = [mag_err_trans1, mag_err_trans2, mag_err_trans3]
mag_err_refs  = [mag_err_refs1, mag_err_refs2, mag_err_refs3]


# In[14]:

#Making functions for errorpropagation
def errorprop(errors):
    errors = np.array(errors)
    return np.sqrt(np.sum(np.array(errors)**2))

def errorprop_for_avg(mag_err_refs):
    return [errorprop(mag_err_refs[i])/len(mag_err_refs[i]) for i in range(len(mag_err_r
"""It says:
[errorprop(mag_err_refs[i])/len(mag_err_refs[i])
            for i in range(len(mag_err_refs))]"""

def errors_for_mag_grbdiff(error_for_avg, mag_err_trans):
    return [np.sqrt(error_for_avg[i]**2 + mag_err_trans[i]**2) for i in range(len(mag_er
"""It says:
    [np.sqrt(error_for_avg[i]**2 + mag_err_trans[i]**2)
     for i in range(len(mag_err_trans))]"""

error_for_avg = errorprop_for_avg(mag_err_refs)
mag_err_trans = errors_for_mag_grbdiff(error_for_avg, mag_err_trans)


# In[16]:

#Getting the errer for the drops
drop_err = (errorprop([mag_err_trans[1], mag_err_trans[0]]),
            errorprop([mag_err_trans[2], mag_err_trans[1]]),
            errorprop([mag_err_trans[2], mag_err_trans[0]]))


# In[17]:

#Getting the mags of the transient and the mean of the ref sources
mags_trans = [table1["mag_psf"][grb_index], table2["mag_psf"][grb_index],
              table3["mag_psf"][grb_index]]
```

```
mags_refs  = [np.mean(table1["mag_psf"][ref_idxs]),
              np.mean(table2["mag_psf"][ref_idxs]),
              np.mean(table3["mag_psf"][ref_idxs])]


# In[18]:

#Calculating the drops
def calc_drop(mags_trans, mags_refs):
    diff = (np.array(mags_refs).astype("float") -
                np.array(mags_trans).astype("float"))
    drops = [diff[1] - diff[0], diff[2] - diff[1], diff[2] - diff[0]]
    return drops

drop = calc_drop(mags_trans, mags_refs)


# In[19]:

#Printing the drops
_ = [print("Drop: ", drop[i], "Error:", drop_err[i]) for i in range(len(drop))]
#The last one is the drop between the first and the last image
```

```
Install brew(Optional):
        /bin/bash -c "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
        echo 'eval "$(/home/linuxbrew/.linuxbrew/bin/brew shellenv)"' >>
/home/flemming/.profile
        eval "$(/home/linuxbrew/.linuxbrew/bin/brew shellenv)"


Install fftw: http://wiki.ipb.ac.rs/index.php/SExtractor_installation

Install ATLAS: sudo apt-get install libatlas-base-dev

Install sextractor:
        git clone https://github.com/astromatic/sextractor.git
        cd sextractor-<version>
        sh autogen.sh
        sudo apt-get install libcfitsio-dev
        ./configure
        sudo make install

        test: type sex in terminal

Install Autophotometer:
        git clone https://github.com/VitalyAstro/autophotometer
        cd autophotometer
        pip install .


Install stdpipe:
        git clone https://github.com/karpov-sv/stdpipe.git
        cd stdpipe
        python setup.py develop


Install HOTPANTS:
        git clone https://github.com/acbecker/hotpants.git
        put in f-common after the flags in the makefile. Do it before trying to
run "sudo make all"
        sudo make all
```