Evolved Supernova Remnants in the Large Magellanic Cloud: A JWST Feasibility Study

Mr James Murphy 17204868

A thesis submitted in partial fulfillment for the degree of Masters of Science

in the

School of Physics
University College Dublin

Supervisor: Dr Patrick Kavanagh

August 2018

GitHub:

https://github.com/jaym878/SNR-Feasibility-Study

I certify that this thesis which I now submit for examination for the award of Master of Science is entirely my own work and has not been taken from the work of others, and to the extent that such work has been cited and acknowledged within the text of my work.

This thesis was prepared according to the regulations for graduate study by research of the University College Dublin and has not been submitted in whole or in part for another award in any other third level institution.

The work reported on in this thesis conforms to the principles and requirements of the UCD's guidelines for ethics in research.

UCD has permission to keep, lend or copy this thesis in whole or in part, on condition that any such use of the material of the thesis be duly acknowledged.

_____ _____

James Murphy Date

"The nitrogen in our DNA, the calcium in our teeth, the iron in our blood, the carbon in our apple pies were made in the interiors of collapsing stars. We are made of starstuff."

-Carl Sagan, Cosmos.

Acknowledgements

The author would like to thank the project supervisor, Dr Patrick Kavanagh for helpful suggestions in writing this paper and for the opportunity to conduct this research.

Contents

Abstract

Mid infrared studies of the sky are notoriously difficult as they require the use of a space observatory due to the IR bands being blocked by the atmosphere. This study aims to look at the feasibility of using the James Webb Space Telescope to observe older supernova remnants in the mid-IR spectrum, as they are dimmer and have not previously been observed with high sensitivity. To accomplish this, a Python package was written which analysed Spitzer images and calculated relevent fluxes. These image characteristics were then used in the MIRI simulation software such as the Exposure Time Calculator (ETC) and the MIRISim. The Python package was written in an object oriented format allowing the compiled catalog to be assessed and evaluated easily. This led to the results being returned in an easy to read format, and allowing for the study to be conducted. The study found that SNRs with dimmer fluxes could be detected and catagorised by JWST. This was accomplished via the use of aperture photometry on each SNR and using the MIRI ETC and MIRISim software suites. The result was an increased number of older SNRs being potentially detected by JWST.

CHAPTER 1

Introduction

Evolved supernova remnants have not yet been studied in detail due to the mid-IR band being blocked by the atmosphere, requiring the use of space telescopes. Current space telescopes do not have the sensitivity to analyse these remnants in great detail. This study uses images from the Spitzer heritage archive along with the MIRI simulator to assess the feasibility of using the James Webb Space Telescope to observe these remnants.

The life-cycle of cosmic dust has previously been explored. However, due to the limitations of IR space telescopes, it has been difficult to explore the cycle in its entirety. Older SNRs are the main point of emphasis in this study as they are much more difficult to observe and may hold the answers cosmologists have to these questions. This report explores the potential of using the JWST MIRI (Mid InfraRed Instrument) to investigate the life cycle of interstellar dust in the shockwave of an evolved supernova remnant. To this end, a Python package was written to download and analyse Spitzer images as potential targets for JWST. A catalogue of potential sources was compiled using previously published papers resulting in 63 targets being identified in the LMC (Large Magellanic Cloud) to be used as a basis for this study. The results from the Spitzer analysis were then used in the JWST Exposure Time Calculator and the MIRI Simulator to obtain a required observation time via a high signal to noise ratio. The MIRI Simulator also returns false colour images corresponding to what specific MIRI filters will see. The feasibility of this study is the main emphasis of this report.

CHAPTER 2

Theory

2.1 Supernova Remnants

A supernova remnant SNR is the object which results from the interaction of supernova ejecta with the interstellar medium (ISM). Supernovae are the brightest phenomena in the universe due to the massive amounts of energy released by the star during the explosion. This released energy is on the order of 10^{51}erg. The explosion shockwaves can have velocities up to 10% of the speed of light. This shockwave travels through space, impacting and heating the interstellar medium. This effect can be detected from the γ-ray to the radio spectrum through a variety of mechanisms [1].

A supernova remnant undergoes 4 stages in its life cycle:

1. Free expansion of the ejecta.
2. Pressure driven expansion via the Sedov-Taylor blast wave.
3. Momentum driven snowplough phase.
4. Merging with the surrounding interstellar medium.

Due to the large amount of dust with temperatures between 100-500k, the visible spectrum lies in the mid IR band. The peak flux is now in the mid IR band making a space observatory a requirement. However, modern observatories struggle to see the faintest of SNRs due to background contamination and spectral resolution [1].

2.2 Large Magellanic Cloud

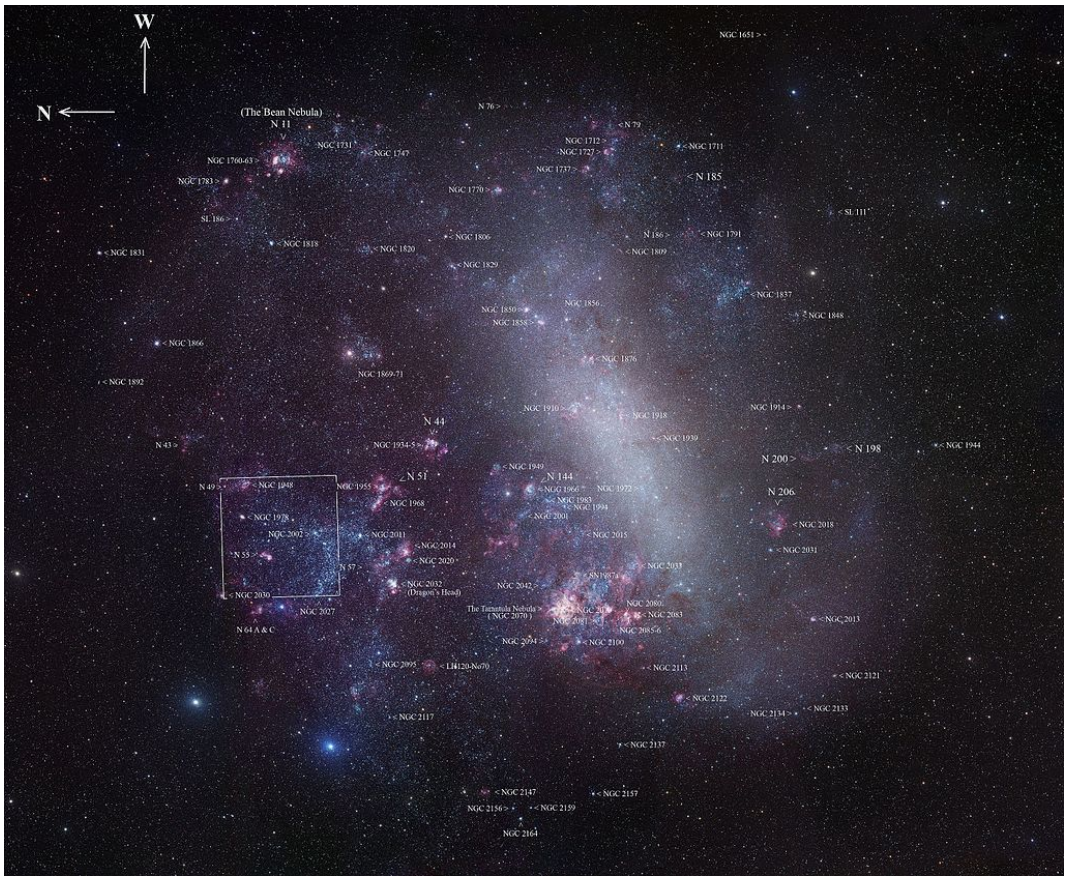

Figure 2.1: Map of the Large Magellanic Cloud. Note the large number of nebulae in the Eastern section also known as the Tarantula nebula. Credit: ESO.

The Large Magellanic Cloud (LMC) is a dwarf galaxy located in the Dorado and Mensa constellations. The LMC is approximately 50kpc away from the Earth. It is located outside the plane of the Milky Way which reduces the amount of background noise created by our Galaxy when looking for SNRs, making the LMC the best candidate for searching for dimmer SNRs.

It has many previously catalogued SNRs, giving a large base of data to work with. The LMC also contains the most recent supernova in the local group, SNR 1987A, which was also included in the catalogue used by this study.

Previous surveys of the LMC have also been conduced in the X-ray and IR bands by ESA's XMM Newton (X-ray) and NASA's Spitzer Space Telescope (IR), giving a wealth of data for use in the analysis.

2.3 Interstellar Medium

Dust, or the interstelar medium (ISM), plays an important role in all stages of galaxy evolution, as its life cycle and the amount and relative abundances present in the ISM are determined by the balance between grain formation, modification, and destruction. The dust heated in the powerful shock waves of SNRs radiates most strongly in the mid IR waveband. Therefore, space based IR observations provide the only means to study both the properties of ISM dust and the formation and destruction mechanisms. However, the full life cycle of an SNR is not entirely known. The later life of an SNR is currently out of the sensitivity range of current telescopes such as Spitzer. JWST presents an opertunity to study these older remnants as the sensitivity of JWST is much higher than previous missions. This new insight may provide the conclusion to the theory surrounding the life cycle of a supernova [2].

2.4 Spitzer Space Telescope

2.4.1 Overview

Figure 2.2: Image of NASA's Spitzer space telescope. Credit: NASA.

The Spitzer space telescope was launched in 2003 with the primary mission of observing the sky in the mid and near IR bands (3.6μm to 160μm). The planned mission period was to be 2.5 years with a pre-launch expectation that the mission could extend to five or slightly more years until the onboard liquid helium supply was exhausted. Without liquid helium to cool the telescope to the very low temperatures needed to operate, most of the instruments are no longer usable. However, the two shortest-wavelength modules of the IRAC camera are still operable with the same sensitivity as before the cryogen was exhausted, and have continued to be used 15 years on.The Spitzer space telescope carries 3 instruments on board. IRAC, IRS and MIPS. However, this study looks primarily at IRAC and MIPS as the study did not require a spectrograph [8].

The Spitzer Agents of Galactic Evolution SAGE survey was used as the primary source of images in this study. The SAGE survey focused on the key transition phases of matter are traced via dust emission in the interstellar medium with a focus on the LMC. The emission from the coldest dust was also traced by the Herschel Observatory in the imaging program HERschel Inventory of The Agents of Galaxy Evolution (HERITAGE) in both Magellanic Clouds [4].

2.4.2 Instruments

InfraRed Array Camera (IRAC)

The InfraRed Array Camera or IRAC operates 4 different filters and imagers via the 3.6μm, 4.5μm, 5.8μm and 8.0μm wavebands. Each filter has a 256x256 pixel imager and a pixel size of 3cm. The IRAC imager observes 2 fields simultaneously with centres 6.5 arcmin apart leaving a gap of 1.5 arcmin between the images. The two short wavelength channels use InSb detector arrays and the two longer wavelength channels use Si:As detectors [8].

Multiband Imaging Photometer for Spitzer (MIPS)

The Multiband Imaging Photometer for Spitzer or Spitzer or MIPS consists of 3 detectors in the far IR with wavebands and pixel densities of 128x128 at 24μm, 32x32 at 70μm and 2x2 at 160μm. All three imagers view the sky simultaneously. The 24 micron camera has a 5x5 arcminute FOV as does the 70 micron, but due to a cabling problem, this is reduced to 2.5x5 arcmin. The 160 micron has a FOV of 0.5x5 arcmin. MIPS has four basic operating modes and associated Astronomical Observation Templates AOT. The Scan Mapping AOT is used to image large areas of the sky in several bands. The Photometry and Super Resolution AOT is used to image sources smaller than 2 arcmin diameter, including point sources. The Spectral Energy Distribution (SED) AOT is used to obtain low-resolution spectra in the 70 micron band. The Total Power (TP) AOT is used to measure the absolute brightness of highly extended emission [8].

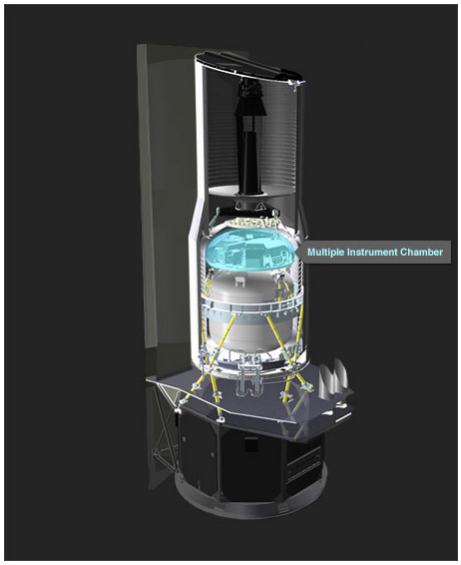

Figure 2.3: Diagram of Spitzer's Multiple Instrument Chamber where IRAC and MIPS are located. Credit: JPL.

2.5 James Webb Space Telescope

2.5.1 Overview

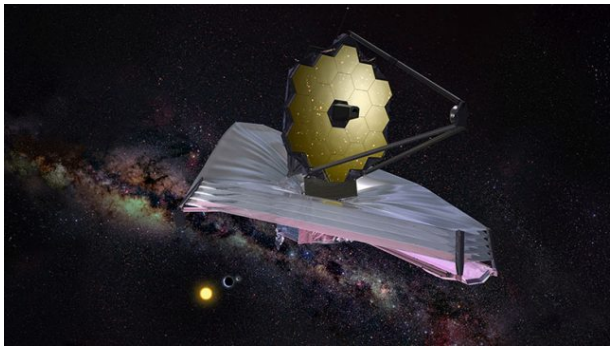

Figure 2.4: Image of the James Webb Space Telescope (JWST) [9].

The James Webb Space Telescope is the spiritual to be the successor to the Hubble Space Telescope and as it is designed to observe in the near and mid IR bands, it can also be considered to be the successor to Spitzer. JWST contains 4 instruments, NIRCam, NIRSpec, MIRI and NIRISS. JWST will surpass Spitzer's capabilities in every way. The primary mirror on JWST is 6.5m compared to Spitzer's 0.85m, resulting in much better spatial resolution. The range of filters is also much larger on MIRI compared to the IRAC instrument on Spitzer, allowing for a more accurate representation of a spectral energy distribution. The detector array is also 4 times larger on JWST's MIRI, with a pixel density of 1024x1024 [9].

2.5.2 Mid InfraRed Instrument MIRI

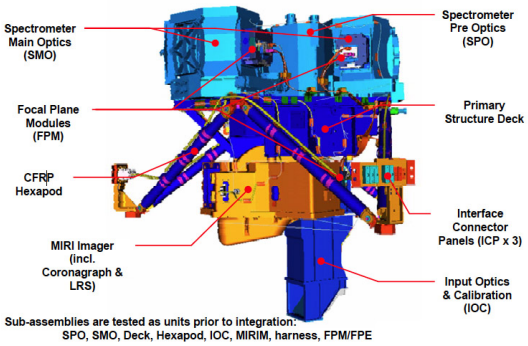

Figure 2.5: Diagram of the Mid InfraRed Instrument on board JWST.

MIRI is both a camera and spectrograph which observes the mid IR band between 4.6μm and 28.6μm with an array 1024x1024 pixels. The imager is designed for larger fields of view, whereas, the spectrograph is designed to sample a much smaller area of the sky [9].

CHAPTER 3

Methodology

3.1 Catalogue

A catalogue containing a list of potential targets was compiled from [3] which is the most complete list and is based on X-ray observations. The catalogue contained information available on each source. Each source had previous information on their X-ray luminosity from the VizieR database [7]. Other characteristics such as their IR luminosities were taken from the list compiled in [5]. Several required characteristics needed to be calculated. [3] paper gave values on radius in degrees across the sky, which needed to be converted to parsec using the equation below

$$R_{pc} = \tan(\theta_{radians})5e10^4 \tag{3.1}$$

The shockwave velocity was also calculated from the estimated plasma temperatures (kT) given in the VizieR database. This was calculated using

$$V_{sh} = \sqrt{\frac{16kT}{3\mu}} \tag{3.2}$$

where μ is the mean mass per particle. The age was then calculated using the formula

$$Age = \frac{2R}{5VS_{sh}} \tag{3.3}$$

The catalogue was then passed to the user file in the Python package where it was read in. Example lines of the input catalogue are shown in Table3.1.

Table 3.1: Example of the input catalogue

MCSNR	RAJ2000	DEJ2000	L_x	Radius	kT	V_{sh}	Age
Name	hr	deg	erg	pc	keV	m/s	kyr
J0448-6700	04 48 22	-66 59 52	4.6E+33	26.67	0.21	4.2E+5	24.87
J0449-6920	04 49 20	-69 20 20	7E+32	19.64	0.2	4.1E+5	18.77
J0450-7050	04 50 27	-70 50 15	5.9E+33	41.21	0.24	4.5E+5	35.95
J0453-6655	04 53 14	-66 55 13	1.17E+34	31.03	0.36	5.5E+5	22.1
J0453-6829	04 53 38	-68 29 27	1.39E+35	14.55	0.37	5.6E+5	10.22

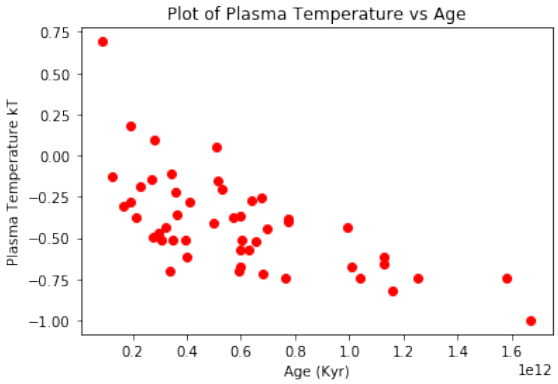

Figure 3.1: Plot of Plasma Temp vs Age

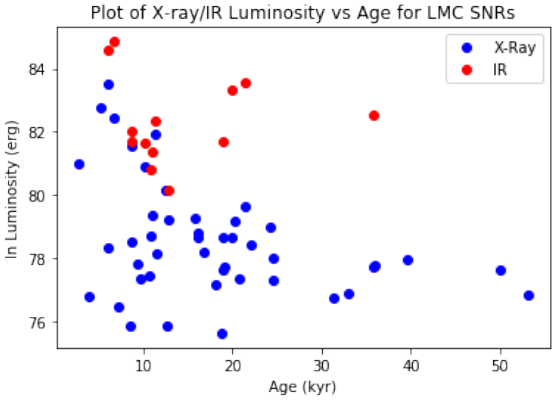

Figure 3.2: Plot of Luminosities vs Age.

These figures show data from previous studies such as XMM-Newton and Spitzer, plotted together.

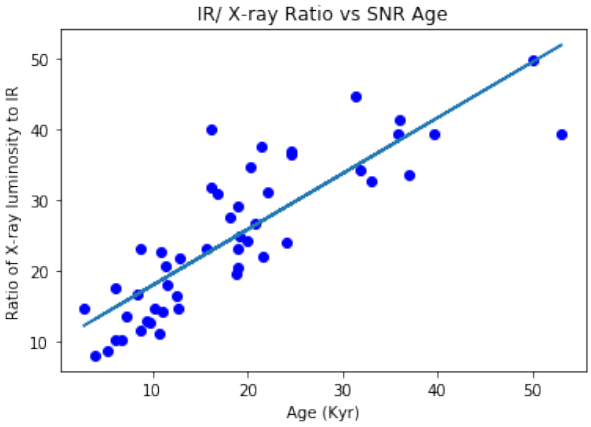

Figure 3.3: This plot shows the relationship between the X-ray luminosity and the IR fluxes of the targets in the LMC. Note how the ratio increases in older SNRs showing the prevalence of IR in this stage.

3.2 Code Overview

A package was written in Python to load the data from the catalogue and analyse it. This involved writing a Python class for usability, and a specialised Python program. A flowchart of how the package handles data is shown below.

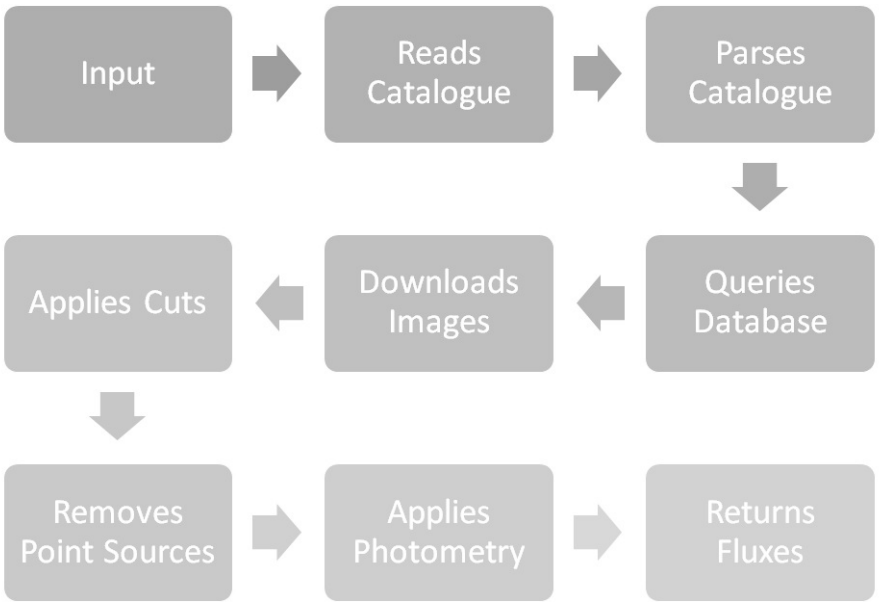

Figure 3.4: caption

The catalogue is read in by the snr_analysis.py script, where the variables are parsed and ordered into a format which could be used by the python package astropull.py and its ImagePull class.

3.2.1 User Program

The package created for this study was designed with re-usability in mind. To this end, a user script was written specifically for this study, allowing the class to be used in other applications. The user script (snr_analysis.py) worked specifically with the catalogue mentioned earlier. This allowed tailoring of the program to read in and pass data in a format applicable to the target catalogue. The user program was also written to save all passed data from the class such as the returned fluxes in both flux and flux density. The idea of re-usability was reiterated here in how each set of data was saved. Several .txt files containing the averaged fluxes and flux densities around the centre of the SNR and across its blst wave in the form of an annulus, were created in the target folder. These fluxes were then plotted vs wavelength to show the spectral energy distribution. This plot was also saved in the target folder.

In summary, the user program iterates over the entire catalogue, applying the analysis to each target and passing its variables to the created package, allowing for increased usability while reducing overall runtimes.

3.2.2 Python Package

Due to the requirement for re-usability, the majority of the project was written using object oriented Python. This led to the creation of an independent Python package (astropull.py) where the class (ImagePull) was defined. The ImagePull class contains 8 individual methods along with an __init__ and run method.

1. __init__: This method defined the variable which were being passed from the user file to the class.
2. eq2deg: Converted hms/dms to decimal degrees.
3. query: This method created a skycoord reference which was used when it referenced the Spitzer Heritage Archive. This then downloaded mosaic images in the referenced waveband.
4. cuts: This method only downloads wanted images from IRAC and removes unwanted images from MIPS.
5. filled sources: This method detects and removes point sources by creating a mask and filling the gaps using the photutils package.
6. plot_image: This is a convenience function to plot data on a projection of the world coordinate system (wcs) and saves the plot as a .pdf.
7. show_image: This method displays the plotted image for the user while the code is running.

8. ap_phot: This method applies aperture pohotometry over the entire area of the SNR, returning an overall average, corrected for the MIRI imager FOV.

9. flux_arc: This method applies aperture photometry around the annulus of the SNR and returns the averaged flux.

10. run: This method runs all other methods within the class. This allows the user file to run the entire class by just calling the run method in ImagePull.

3.2.3 Required Packages

The ImagePull class requires 3 aditional non standard python packages, astropy, photutils and astroquery. These packages can be downloaded from the online python library using the pip install or directly from the repositories.

3.3 Simulation

3.3.1 JWST Exposure Time Calculator

The JWST Exposure Time Calculator (ETC) is a pixel-based ETC paired with a modern graphical user interface. It supports all JWST observing modes: imaging, spectroscopy (slitted, slitless, and IFU), coronagraphy, and aperture masking interferometry. The ETC was created to allow users to upload fluxes and determine signal to noise ratios. To accomplish this, the ETC allows the user to have multiple workbooks. The user must define sources, place them in scenes, and use the scenes in calculations. Sources may be used in multiple scenes, and scenes may be used by multiple calculations, which will be automatically recalculated to reflect any changes made. The result of the ETC is an exposure time and signal to noise ratio, which can then be used in the MIRI simulator [10].

3.3.2 MIRI Simulator

The goal of the MIRISim is intended to allow future MIRI users to gain experience with the data produced by the instrument, how MIRI data is reduced by the JWST development pipeline, and help with observation feasibility studies. MIRISim generates data files as uncalibrated JWST exposure and returns data as .fits files. The MIRISim simulates the Low Resolution Spectrograph (LRS), Medium Resolution Spectrograph (MRS) and Imgager main field [11].

CHAPTER 4

Results

4.1 Overview

4.2 Case Study - DEM L205 (J0528-6727)

4.2.1 Spitzer Image Analysis

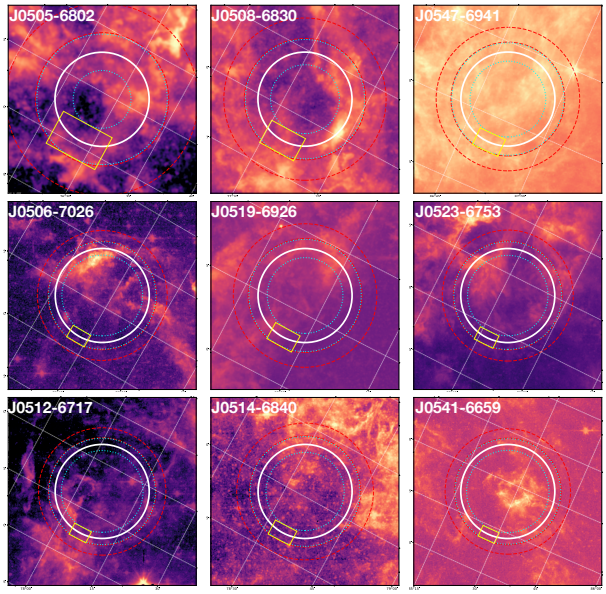

Figure 4.1: ESpitzer IRAC 8.0 micron images of a selection of SNRs. The top row shows SNRs with ages less that 10 kyr, the middle row are 10-20 kyr, and the bottom row ar 20-30 kyr.

DEM L205 was chosen due to its older age and low X-ray luminosity taken from the XMM-Newton survey. This SNR was also one of the SNRs in the catalog to have all data available for this study. This allowed a complete overview of a target which had the expected low IR flux and with a potential peak of flux in the expected waveband.

IRAC Images

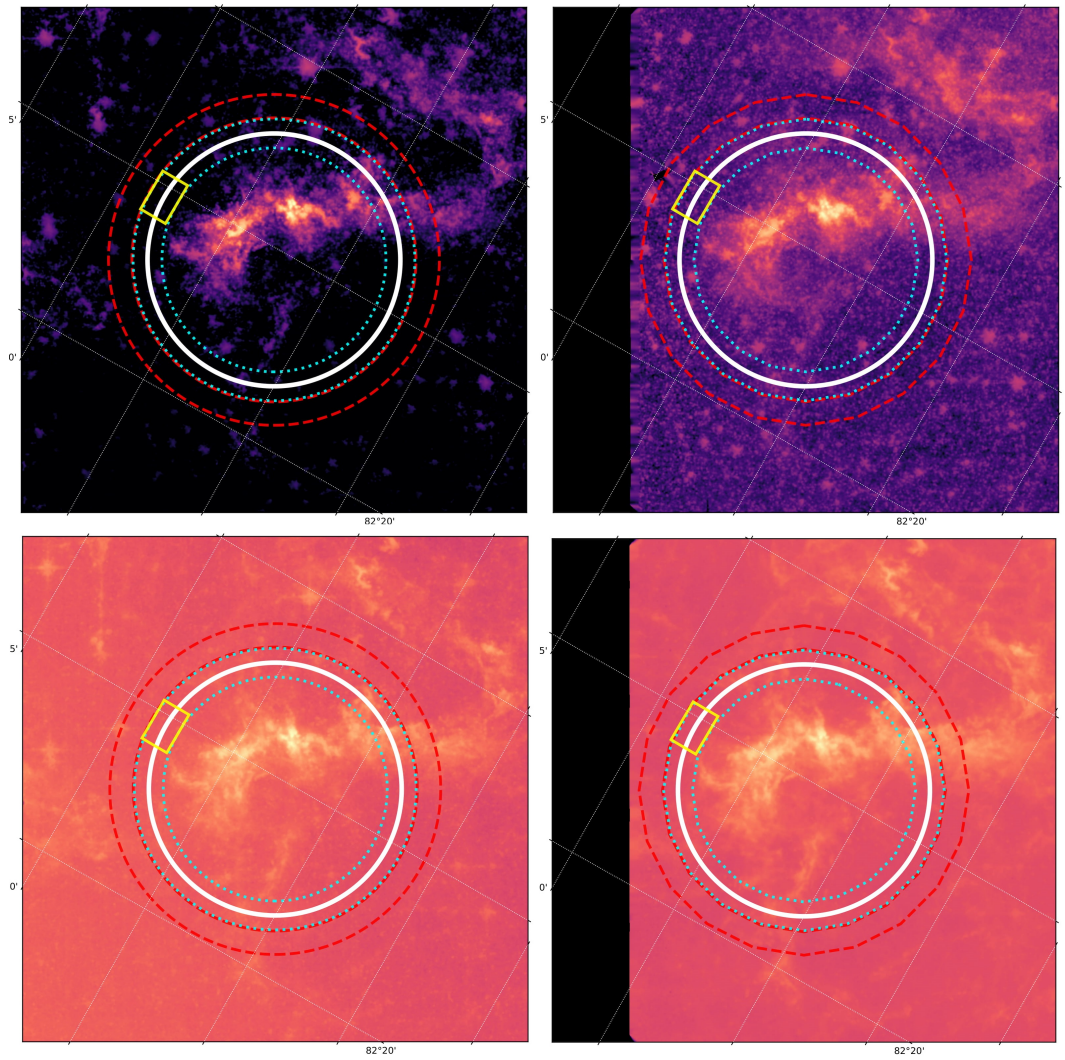

Figure 4.2: Plotted IRAC images uring the 3.6μm filter (top left), 4.5μm filter (top right), 5.8μm filter (bottom left), 8.0μm filter (bottom right).

Figure 4.2 displays the IRAC images of DEM L205. These images are colour coded to show how thermal emission and background contamination becomes an issue at longer wavelengths. These images also display how aperture photometry was conducted on the image. The solid white circle denotes the calculated radius of the SNR taken from the input catalogue. The yellow box denotes the MIRI FOV and the cyan circles denote the annulus where the photometry was conducted. This annulus corresponds to the MIRI FOV located where the SNR is brightest,on the shockwave, giving the upper limit estimate of the SNR's flux. Background was also accounted for by applying aperture photometry to the darker regions surrounding the SNR (greater than its radius) and subtracting the average flux found here from the average flux inside the SNR. This is denoted by the red circles.

MIPS Images

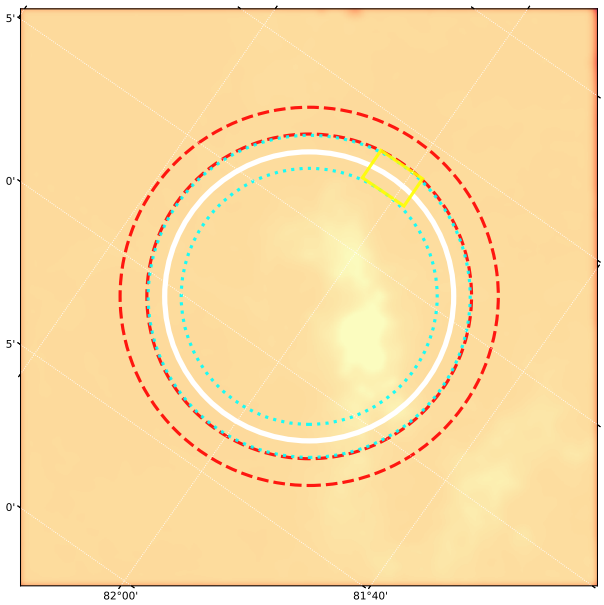

Figure 4.3: 24μm MIPS filter.

Thermal emission becomes a larger issue with the MIPS images as these are taken in the far infrared. The 24μm filter can still be considered applicable data for this study as the source is not yet overshadowed by background contamination. The photometry in the MIPS images is conducted in the same format as the IRAC images. However, due to the pixel scale being on a different order, the flux was calculated using their own pixel scales mentioned in the MIPS theory section.

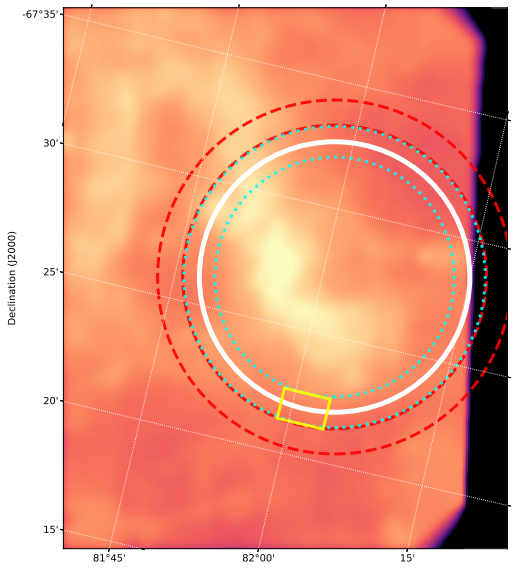

Figure 4.4: 70μm MIPS filter.

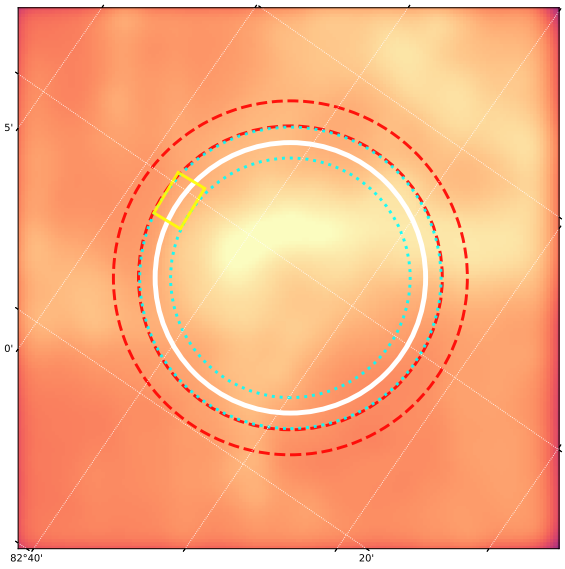

Figure 4.5: 160μm MIPS filter.

Source Removal

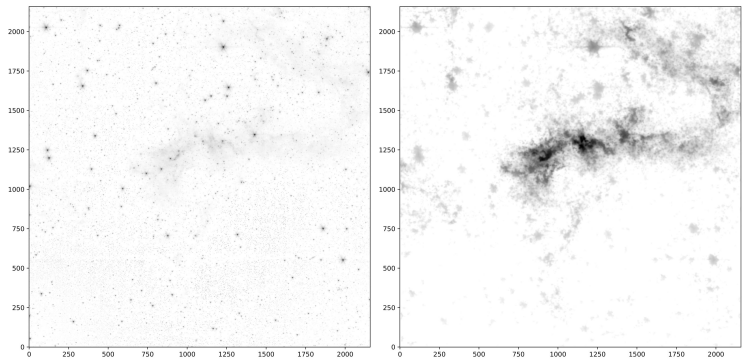

Figure 4.6: This image displays a raw (left) black and white image of DEM L205 and the same image with point sources removed (right).

The sources were removed by a method in the ImagePull class which identified point sources and placed a mask of equivalent flux from the surrounding background over the source. This was done to reduce background contamination in the flux calculations.

4.2.2 MIRI Simulator Results

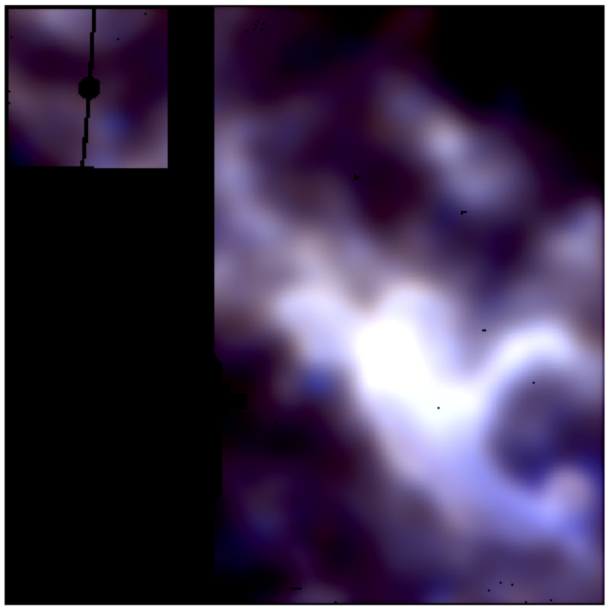

Figure 4.7: False-color image of part of an SNR shell (DEM L205) produced using the MIRI Simulator with exposure times set using the ETC.

The RGB in the simulated image in Figure 4.7 corresponds to the 5.6, 7.7, and 10.0 micron MIRI Imager filters. Note that JWST/MIRI spatial resolution will be much better and the shell looks smooth here because the simulator resamples lower resolution Spitzer data.

4.2.3 DEM L205 Results

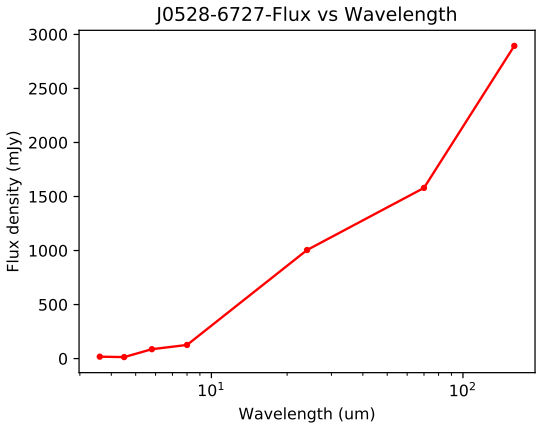

Figure 4.8: Plot of DEM L205 flux density vs wavelength

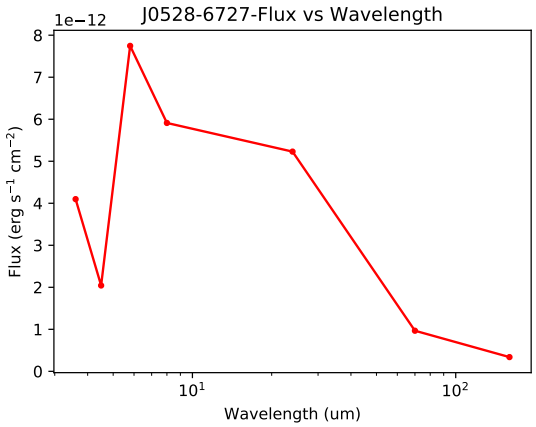

Figure 4.9: Plot of DEM L205 fluxes vs wavelength.

The plot of flux density (left) shows how thermal emission dominates the longer wavelengths. This shows how the far IR band is not an ideal observational environment for SNRs. The flux vs wavelength (right) plot shows how there is a clear emission peak in the 8μm filter. This flux could then be directly compared to the future MIRI 7.7μm filter to determine an upper limit.

4.3 Feasibility

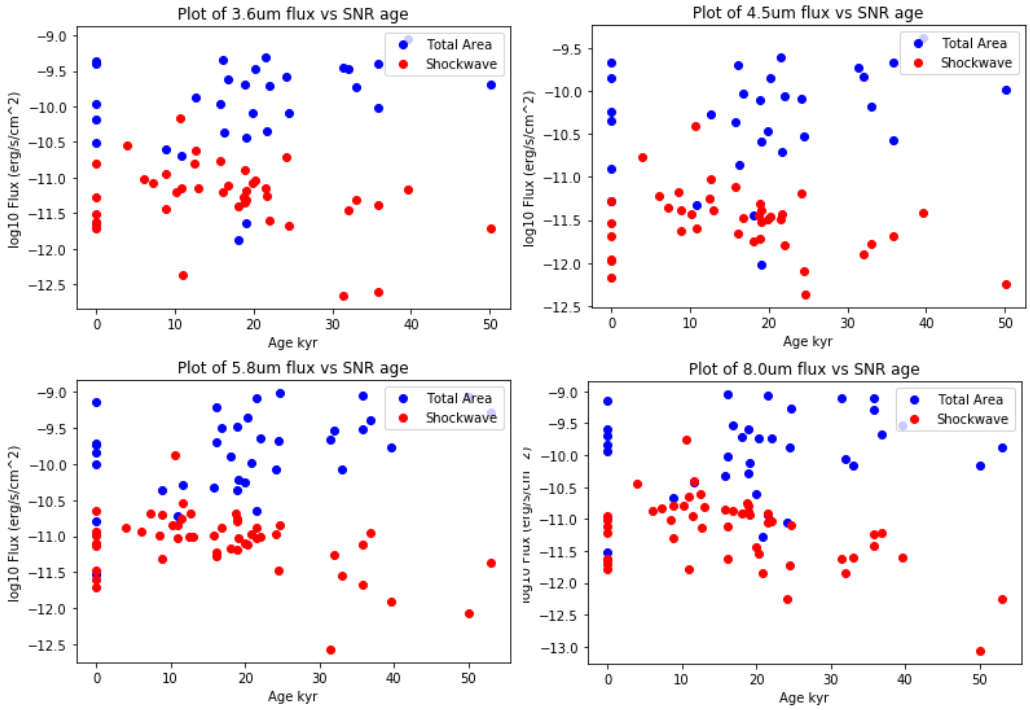

Figure 4.10: Plots of flux vs age for all SNR targets.

These plots show how the fluxes of the shockwave are almost always higher than the total average of the SNR, showing how and why the upper level of the flux was calculated and used in these calculations.

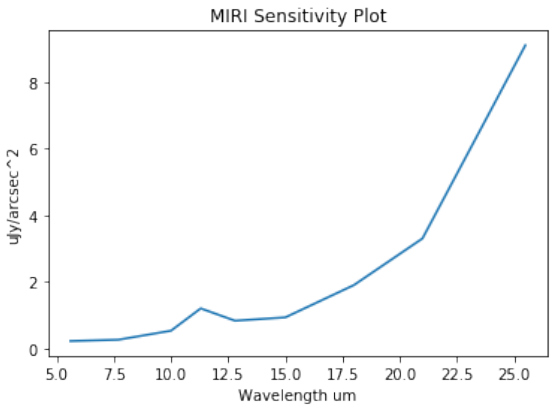

Figure 4.11: MIRI Sensitivity

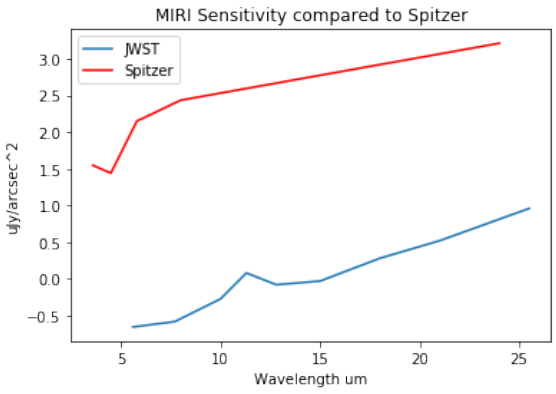

Figure 4.12: MIRI Sensitivity compared to Spitzer

The diference between the sensitivities of MIRI and Spitzer's shorter wavebands is shown in Figure 4.12. This is a log graph as the differences of sensitivity is quite large, showing how JWST outclasses Spitzer in every way. This difference can then be used to extrapolate how many SNRs below the Spitzer sensitivity can be seen by MIRI.

CHAPTER 5

Discussion of Results

5.1 Spitzer Results Discussion

The package created for this study worked exactly as designed. The entire list of 63 SNRs were analysed and data was collected on every filter in IRAC and MIPS this data was then plotted to show the both the flux and flux densities of each SNR and for each filter overall. It was also shown in Figure 4.9 how thermal emission overcomes data in longer wavelength images which allows this study to discount the 70μm and 160μm filters from MIPS. The remaining filters were then compared to one another to find which waveband would be best to be used by MIRI. The plot in Figure 4.9 shows a clear peak at 8.0μm for DEM L205. This is a characteristic shared by most, but not all SNRs in the catalogue.

5.2 MIRI Simulator Results Discussion

The Exposure Time Calculator was run for DEM L205 with it's given fluxes were determined. This allowed the characteristics for the 8.0μm filter to be fully simulated in MIRISim. The resulting image gave a false colour image of the shell of DEM L205 with several MIRI wavebands. The image in Figure 4.7 displays a simulated image created from limited hardware aboard Spitzer. Therefore, the spatial resolution will be much better when taken from JWST. The shel also looks smoother as the simulator simply resamples lower resolution Spitzer data.

CHAPTER 6

Conclusion

We have created a Python package which references the Spitzer heritage archive and fetches images in all wavebands observed by Spitzer. The package then analyses these images and returns expected fluxes which will be seen by JWST's MIRI instrument. This is then used in MIRI simulation software to assess the feasibility of observing the faintest SNRs in the LMC. The Python package written worked just as designed and allowed an easy evaluation of potential candidates alongside the compiled catalogue. The Python package was written in an object oriented format allowing the compiled catalog to be assessed and evaluated easily. This led to the results being returned in an easy to read format, allowing for the study to be conducted. The study found that SNRs with dimmer fluxes could be detected and catagorised by JWST. This was accomplished via the use of aperture photometry on each SNR and using the MIRI ETC and MIRISim software suites. The plot in Figure 4.9 show at clear peak in flux at 8μm which would correspond to the MIRI 7.7 μm. The plots in Figure 4.10 displays how the upper limits in every target was found around the annulus of the shockwave. This gave the best case scenario for the sensitivity calculations. The overall result was an increased number of older SNRs being potentially detected by JWST.

CHAPTER 7

Appendix

7.1 User Program (snr_analysis.py)

```
from astropull import ImagePull
from astropy import units as u
import numpy as np
import os

import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

from astropy.table import Table

def arrays(pos, line):
    """
    I think this function can be separated as it parses your catalogue
    independent of catalogue formats, etc., so best to get the various
    ImagePull.
    """
    list = []
    num = line.shape[0]
    for n in range(num):
        list.append(line[n][pos])
    list = np.asarray(list)

    return list
```

```
def parse_cat_file(filename, src_num):
    """
    Wrap the fetching of columns from the file in a function
    Input
    ======================
    filename (str):
        catalogue filename
    src_num (int):
        the number of the source in the catalogue
    """
    # Loads csv file
    line = np.genfromtxt(open(filename, "r"), names=True, delimiter=','

    # Arrays Function
    # MCSNR
    MCSNR = arrays(0, line)
    MCSNR = MCSNR[src_num]
    MCSNR = MCSNR.decode("utf-8")

    # RA
    RA = arrays(1, line)
    RA = RA[src_num]

    # DE
    DE = arrays(2, line)
    DE = DE[src_num]

    # Radius
    Rad = arrays(8, line)
    Rad = Rad[src_num]

    # kT
    kT = arrays(12, line)
    kT = kT[src_num]

    # VShock
    VShock = arrays(16, line)
    VShock = VShock[src_num]
```

```
    # Age
    Age = arrays(18, line)
    Age = Age[src_num]

    # LX
    LX = arrays(4, line)
    LX = LX[src_num]

    # LIR
    LIR = arrays(7, line)
    LIR = LIR[src_num]

    return MCSNR, RA, DE, Rad, kT, VShock, Age, LX, LIR


# setup master output array
col_names = ('name', 'radius', 'kT', 'v_s', 'age', 'Lx',
             '3.6_flux', '3.6_arc_flux', '3.6_Jy', '3.6_arc_Jy',
             '4.5_flux', '4.5_arc_flux', '4.5_Jy', '4.5_arc_Jy',
             '5.8_flux', '5.8_arc_flux', '5.8_Jy', '5.8_arc_Jy',
             '8.0_flux', '8.0_arc_flux', '8.0_Jy', '8.0_arc_Jy',
             '24_flux', '24_arc_flux', '24_Jy', '24_arc_Jy',
             '60_flux', '60_arc_flux', '60_Jy', '60_arc_Jy',
             '120_flux', '120_arc_flux', '120_Jy', '120_arc_Jy')

col_dtype = ('S10', 'f4', 'f4', 'f4', 'f4', 'f4',
             'f4', 'f4', 'f4', 'f4',
             'f4', 'f4', 'f4', 'f4',
             'f4', 'f4', 'f4', 'f4',
             'f4', 'f4', 'f4', 'f4',
             'f4', 'f4', 'f4', 'f4',
             'f4', 'f4', 'f4', 'f4',
             'f4', 'f4', 'f4', 'f4')

master_output = Table(None, names=col_names, masked=True, dtype=col_dty

# determine number of SNRs in catalogue
n_snrs = len(np.genfromtxt(open('SNR_list.csv', "r"), names=True, delim
```

```
# get SNR properties using ImagePull
for SNR in range(n_snrs):

    MCSNR, RA, DE, Rad, kT, VShock, Age, LX, LIR = parse_cat_file('SNR

    print("\nRunning {}\n".format(str(MCSNR)))

    # write the known parameters to master output
    snr_row = [MCSNR, Rad, kT, VShock, Age, LX, 0, 0, 0, 0, 0, 0, 0, 0,
               0, 0, 0, 0,0, 0, 0, 0,0, 0, 0, 0]

    # instrument things
    instrument_list = ['IRAC', 'MIPS']
    arc_flux_plot = []
    arc_flux_plot_jy = []

    # Select Instrument and Band
    for sensor in instrument_list:
        flux = []
        arc_flux = []
        flux_jy = []
        arc_flux_jy = []

        if sensor == 'IRAC':
            band_list = [3.6, 4.5, 5.8, 8.0]
            for b in band_list:
                try:
                    # Run image pull as object
                    my_test = ImagePull(MCSNR, RA, DE, Rad, sensor, b)
                    f_jy, f_erg, farc_jy, farc_erg = my_test.run()

                    flux.append(f_erg)
                    arc_flux.append(farc_erg)
                    arc_flux_plot.append(farc_erg)
                    flux_jy.append(f_jy)
                    arc_flux_jy.append(farc_jy)
                    arc_flux_plot_jy.append(farc_jy)

                    if b == 3.6:
```

```
                snr_row[6] = f_erg
                snr_row[7] = farc_erg
                snr_row[8] = f_jy
                snr_row[9] = farc_jy
            if b == 4.5:
                snr_row[10] = f_erg
                snr_row[11] = farc_erg
                snr_row[12] = f_jy
                snr_row[13] = farc_jy
            if b == 5.8:
                snr_row[14] = f_erg
                snr_row[15] = farc_erg
                snr_row[16] = f_jy
                snr_row[17] = farc_jy
            if b == 8.0:
                snr_row[18] = f_erg
                snr_row[19] = farc_erg
                snr_row[20] = f_jy
                snr_row[21] = farc_jy

        except Exception as e:
            print('{}-{}um failed '.format(sensor, b))
            print('  %s: %s' % (e.__class__.__name__, str(e)))

            try:
                os.mkdir(str(MCSNR))
            except:
                pass

            flux.append(np.nan)
            arc_flux.append(np.nan)
            arc_flux_plot.append(np.nan)
            flux_jy.append(np.nan)
            arc_flux_jy.append(np.nan)
            arc_flux_plot_jy.append(np.nan)

            if b == 3.6:
                snr_row[6] = np.nan
                snr_row[7] = np.nan
```

```
            snr_row[8] = np.nan
            snr_row[9] = np.nan
        if b == 4.5:
            snr_row[10] = np.nan
            snr_row[11] = np.nan
            snr_row[12] = np.nan
            snr_row[13] = np.nan
        if b == 5.8:
            snr_row[14] = np.nan
            snr_row[15] = np.nan
            snr_row[16] = np.nan
            snr_row[17] = np.nan
        if b == 8.0:
            snr_row[18] = np.nan
            snr_row[19] = np.nan
            snr_row[20] = np.nan
            snr_row[21] = np.nan



band_list_arr = np.asarray(band_list)
flux_arr = np.asarray(flux)
flux_arc_arr = np.asarray(arc_flux)
flux_jy_arr = np.asarray(flux_jy)
flux_arc_jy_arr = np.asarray(arc_flux_jy)

flux_out = np.column_stack((band_list_arr, flux_arr))
flux_arc_out = np.column_stack((band_list_arr, flux_arc_arr

flux_jy_out = np.column_stack((band_list_arr, flux_jy_arr))
flux_arc_jy_out = np.column_stack((band_list_arr, flux_arc_

out_name = str(MCSNR) + '/fluxes_' + str(MCSNR) + '_' + str
np.savetxt(out_name, flux_out, delimiter=',')

out_name = str(MCSNR) + '/arc_fluxes_' + str(MCSNR) + '_' +
np.savetxt(out_name, flux_arc_out, delimiter=',')

out_name = str(MCSNR) + '/fluxes_Jy_' + str(MCSNR) + '_' +
```

```
        np.savetxt(out_name, flux_jy_out, delimiter=',')

        out_name = str(MCSNR) + '/arc_fluxes_Jy_' + str(MCSNR) + '_
        np.savetxt(out_name, flux_arc_jy_out, delimiter=',')


    if sensor == 'MIPS':
        band_list = [24, 70, 160]
        for b in band_list:
            try:
                # Run image pull as object
                my_test = ImagePull(MCSNR, RA, DE, Rad, sensor, b)
                f_jy, f_erg, farc_jy, farc_erg = my_test.run()

                flux.append(f_erg)
                arc_flux.append(f_erg)
                arc_flux_plot.append(farc_erg)
                flux_jy.append(f_jy)
                arc_flux_jy.append(farc_jy)
                arc_flux_plot_jy.append(farc_jy)

                if b == 24:
                    snr_row[22] = f_erg
                    snr_row[23] = farc_erg
                    snr_row[24] = f_jy
                    snr_row[25] = farc_jy
                if b == 70:
                    snr_row[26] = f_erg
                    snr_row[27] = farc_erg
                    snr_row[28] = f_jy
                    snr_row[29] = farc_jy
                if b == 160:
                    snr_row[30] = f_erg
                    snr_row[31] = farc_erg
                    snr_row[32] = f_jy
                    snr_row[33] = farc_jy

            except Exception as e:
                print('{}-{}um failed '.format(sensor, b))
```

```
            print('  %s: %s' % (e.__class__.__name__, str(e)))

            try:
                os.mkdir(str(MCSNR))
            except:
                pass

            flux.append(np.nan)
            arc_flux.append(np.nan)
            arc_flux_plot.append(np.nan)
            flux_jy.append(np.nan)
            arc_flux_jy.append(np.nan)
            arc_flux_plot_jy.append(np.nan)

            if b == 24:
                snr_row[22] = np.nan
                snr_row[23] = np.nan
                snr_row[24] = np.nan
                snr_row[25] = np.nan
            if b == 70:
                snr_row[26] = np.nan
                snr_row[27] = np.nan
                snr_row[28] = np.nan
                snr_row[29] = np.nan
            if b == 160:
                snr_row[30] = np.nan
                snr_row[31] = np.nan
                snr_row[32] = np.nan
                snr_row[33] = np.nan



band_list_arr = np.asarray(band_list)
flux_arr = np.asarray(flux)
flux_arc_arr = np.asarray(arc_flux)
flux_jy_arr = np.asarray(flux_jy)
flux_arc_jy_arr = np.asarray(arc_flux_jy)

flux_out = np.column_stack((band_list_arr, flux_arr))
```

```
            flux_arc_out = np.column_stack((band_list_arr, flux_arc_arr

            flux_jy_out = np.column_stack((band_list_arr, flux_jy_arr))
            flux_arc_jy_out = np.column_stack((band_list_arr, flux_arc_

            out_name = str(MCSNR) + '/fluxes_' + str(MCSNR) + '_' + str
            np.savetxt(out_name, flux_out, delimiter=',')

            out_name = str(MCSNR) + '/arc_fluxes_' + str(MCSNR) + '_' +
            np.savetxt(out_name, flux_arc_out, delimiter=',')

            out_name = str(MCSNR) + '/fluxes_Jy_' + str(MCSNR) + '_' +
            np.savetxt(out_name, flux_jy_out, delimiter=',')

            out_name = str(MCSNR) + '/arc_fluxes_Jy_' + str(MCSNR) + '_
            np.savetxt(out_name, flux_arc_jy_out, delimiter=',')

    # add everything to the master output table
    master_output.add_row(snr_row)

    # plot in SED erg/cm2/s
    fig, axs = plt.subplots(1, 1, figsize=(5, 4))

    band_list = [3.6, 4.5, 5.8, 8.0, 24, 70, 160]

    axs.plot(band_list, arc_flux_plot, 'r-', marker='o', markersize=3,
    name = str(MCSNR) + '-Flux vs Wavelength'
    axs.set_title(name)
    axs.set_xlabel('Wavelength (um)')
    axs.set_ylabel(r'Flux (erg s$^{-1}$ cm$^{-2}$)')
    axs.set_xscale('log')
    plt.tight_layout()

    plot_name = os.path.join(str(MCSNR) + '/flux_plot_' + str(MCSNR) +

    try:
        os.remove(plot_name)
    except:
        pass
```

```
    # plot SED in Jy
    fig.savefig(plot_name, dpi=200)

    fig, axs = plt.subplots(1, 1, figsize=(5, 4))

    band_list = [3.6, 4.5, 5.8, 8.0, 24, 70, 160]

    arc_flux_plot_jy_arr = np.asarray(arc_flux_plot_jy)
    axs.plot(band_list, arc_flux_plot_jy_arr * 1.e3, 'r-', marker='o',
    name = str(MCSNR) + '-Flux vs Wavelength'
    axs.set_title(name)
    axs.set_xlabel('Wavelength (um)')
    axs.set_ylabel('Flux density (mJy)')
    axs.set_xscale('log')
    plt.tight_layout()

    plot_name = os.path.join(str(MCSNR) + '/flux_Jy_plot_' + str(MCSNR)

    try:
        os.remove(plot_name)
    except:
        pass

    fig.savefig(plot_name, dpi=200)


# save the master output to file
master_name = 'SNR-catalogue_IR_properties.txt'

try:
    os.remove(master_name)
except:
    pass

master_output.write(master_name, format='ascii')
```

7.2 Python Package (astropull.py)

```
# utilities
#main
import os, glob
from pprint import pprint
from shutil import copyfile, rmtree
from photutils import SkyCircularAperture
from photutils import SkyCircularAnnulus
from photutils import SkyRectangularAperture
from photutils import aperture_photometry
from photutils import datasets

# standard packages
import numpy as np
import math
from matplotlib import pyplot as plt
from matplotlib.colors import LogNorm

# astropy and astroquery
from astroquery import sha
from astropy import coordinates as coord
from astropy import units as u
from astropy.io import fits
from astropy.wcs import WCS
from astropy.coordinates import SkyCoord
from astropy.coordinates import Angle
from astropy.nddata import Cutout2D


class ImagePull:
    """
    ImagePull references the Spitzer Heritage Archive using supplied co
    from both the IRAC and MIPS sensor wavebands. The class then applie
    Write what this class does in the doc string here. Include whatever
    Input
    =========================
    name (str):
        name of the SNR
    ra (float):
        ra of the SNR
```

```
dec (float):
    dec of the SNR
radius (float):
    radius of the SNR in units of arcseconds
sensor (str):
    Spitzer detector (options are IRAC and MIPS)
wavelength (float):
    Spitzer filter required (options are: 3.6, 4.5, 5.8, 8.0, 24.,
Output
==========================
cut images:
    saved pdf images of target in selected wavebands

FluxIR (float):
    flux returned in units of erg/s/cm^2
"""
def __init__(self, name, ra, dec, radius, sensor, wavelength):
    # load input parameters to attributes
    self.name = name
    self.ra = ra
    self.dec = dec
    self.radius = radius
    self.sensor = sensor
    self.wavelength = wavelength

    self.ra_deg = None
    self.dec_deg = None

    self.flux_Jy = None
    self.flux_erg = None


def eq2deg(self, RA, DE):
    """
    eq2deg converts a string of eq coordinates to a decimal degree
    Input
    ==========================
    RA (str):
        ra in units of h/m/s
```

```
        DE (str):
            dec in units of d/m/s
        Output
        ==========================
        im_ra_deg (float):
            ra in units of decimal degrees
        im_dec_deg (float):
            dec in units of decimal degrees
        """
        # debug for function
        x = RA.decode('UTF-8')
        y = DE.decode('UTF-8')

        # Creates Ra and Dec formats from data
        things = x.split()
        Ra = things[0] + 'h' + things[1] + 'm' + things[2] + 's'
        im_ra = Ra

        things = y.split()
        Dec = things[0] + 'd' + things[1] + 'm' + things[2] + 's'
        im_dec = Dec

        a = Angle(im_ra, u.hour)
        im_ra_deg = a.degree

        b = Angle(im_dec)
        im_dec_deg = b.degree

        self.ra_deg = im_ra_deg
        self.dec_deg = im_dec_deg

        return im_ra_deg, im_dec_deg

    def query(self, test_src_coord, sensor, wavelength):
        """
        eq2deg converts a string of eq coordinates to a decimal degree

        Input
        ==========================
```

```
test_src_coord (float):
    coordinates of SNR in decimal degrees

sensor (str):
    sensor name

wavelength (str):
    waveband for sensor

"""
my_query = sha.query(coord=coord.SkyCoord(ra=test_src_coord[0],
                                          unit=(u.degree, u.deg

sensor_l = len(sensor) + 1
sens = sensor.rjust(sensor_l)
name = str(sens) + ' ' + str(wavelength) + 'um'
if sensor == 'MIPS':
    if wavelength >= 100:
        name_l = len(name)
    else:
        name_l = len(name) + 1
    name = name.ljust(name_l)
    #print(name)
mask = (my_query['wavelength'] == name)
if sensor == 'IRAC':
    filesize_mask = (my_query['filesize'] > 2.e8)
    combined_mask = np.logical_and(mask, filesize_mask)
else:
    combined_mask = mask

tbl = my_query[combined_mask]
nimages = len(tbl)
distance_check = np.zeros(nimages)

if sensor == 'IRAC':
    for n, row in enumerate(tbl):
        im_ra = row['ra']
        im_dec = row['dec']
        snr = SkyCoord(test_src_coord[0] * u.degree, test_src_co
```

```
                im_centre = SkyCoord(im_ra * u.degree, im_dec * u.degree
                sep = snr.separation(im_centre)
                distance_check[n] = sep.value

            closest = np.argmin(distance_check)
            my_final_row = tbl[closest]
            url = my_final_row['accessUrl'].strip()
            sha.save_file(url, out_dir=str(self.name) + '_' + str(wavel

        elif sensor == 'MIPS':
            for i in tbl['accessUrl']:
                print("Getting: {}".format(i))
                url = i.strip()
                sha.save_file(url, out_dir=str(self.name) + '_' + str(w

    def fill_sources(self, image, save_plot=True):
        """
        detect sources, mask and fill the gaps using photutils
        Input
        ========================
        image (2D np array)
            image to process
        save_plot (boolean)
            if True, saves a diagnostic plot of pre and post source rem
        Returns
        ========================
        filled_image (2D np array)
            image with sources removed and filled
        """
        from photutils.background import Background2D
        from astropy.stats import gaussian_fwhm_to_sigma
        from photutils import detect_sources
        from astropy.convolution import Gaussian2DKernel, interpolate_re

        # get the background level
        bkg = Background2D(image, 30)
        threshold = bkg.background + (5.0 * bkg.background_rms)

        # create the size of the sources
```

```
sigma = 2.0 * gaussian_fwhm_to_sigma  # FWHM = 2.
kernel = Gaussian2DKernel(sigma, x_size=2, y_size=2)
kernel.normalize()

# search for the sources and create a mask
seg_image = detect_sources(image, threshold, npixels=5, filter_k
source_mask = np.clip(seg_image, 0, 1)

# fill the masked source regions
masked_image = image.copy()
masked_image[source_mask > 0] = np.nan

kernel = Gaussian2DKernel(x_stddev=5)
filled_image = interpolate_replace_nans(masked_image, kernel)

# smooth out hard edges around filled sources
kernel = Gaussian2DKernel(x_stddev=3)
filled_image = convolve(filled_image, kernel)

# plot
if save_plot:
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 8))

    vmin = 0.1
    vmax2 = np.max(filled_image) * 0.9
    ax1.imshow(image, cmap='Greys', origin='lower', norm=LogNorm
    ax2.imshow(filled_image, cmap='Greys', origin='lower', norm=

    # display the plot
    plt.tight_layout()

    # save as a pdf
    plot_name = os.path.join(str(self.name), str(self.name) + '_
                             str(self.wavelength) + '_source_rem
    try:
        os.remove(plot_name)
    except:
        pass
```

```
            fig.savefig(plot_name, dpi=200)

        return filled_image

    def cuts(self, wavelength, sensor, MCSNR):
        """
        cuts only downloads wanted images from IRAC and removes unwante

        Input
        =========================
        MCSNR (str):
            name of SNR

        sensor (str):
            sensor name

        wavelength (str):
            waveband for sensor

        """

        sensor_l=len(sensor)+1
        sens=sensor.rjust(sensor_l)
        name = str(sens) + ' ' + str(wavelength) + 'um'
        my_image_files = glob.glob(os.path.join(str(self.name) + '_' +
        n_images = len(my_image_files)
        counter = 0
        filename = sens[1:] + '_' + str(wavelength) + '.fits'
        array = []
        # cycle through and cut images
        for i in my_image_files:
            with fits.open(i) as hdulist:
                header = hdulist[0].header
                if sensor == 'MIPS':
                    if header['OBJECT'] != 'LMC' or header['OBSRVR'] !=
                        if os.path.exists(os.path.join(my_image_files[c
                            os.remove(os.path.join(my_image_files[count

        for i in range(len(my_image_files)):
```

```
            if not os.path.exists(os.path.join(MCSNR)):
                os.makedirs(os.path.join(MCSNR))
            if os.path.exists(os.path.join(my_image_files[i])):
                copyfile(os.path.join(my_image_files[i]), os.path.join(

        rmtree(os.path.join(str(self.name) + '_' + str(wavelength) + '/

    def plot_image(self, data, wcs, coords=None):
        """
        convience function to plot data on a wcs projection
        Input:
        data -    the data array
        wcs -     the WCS object
        coords -  the coordinates of the object for plotting (1x2 array
        """
        # set up the plot
        fig, axs = plt.subplots(1, 1, figsize=(8, 8), subplot_kw={'proj

        # set up limits of the colour scale for plotting
        vmin = 1e-1
        vmax = np.max(data) * 0.9

        # plot
        axs.imshow(data, cmap='magma', interpolation='nearest', origin=

        #axs.scatter(coords[0], coords[1], transform=axs.get_transform(
        #            edgecolor='white', facecolor='none')

        # define the apertures to plot (these are the same as in other
        # SNR
        position = SkyCoord(self.ra_deg * u.degree, self.dec_deg * u.de
        snr_aperture = SkyCircularAperture(position, r=self.radius * u.
        snr_pix_aperture = snr_aperture.to_pixel(wcs)
        snr_pix_aperture.plot(color='white', lw=5, alpha=1.0)

        # BKG
        r = self.radius * u.arcsec
        r_in = r + (40. * u.arcsec)
        r_out = r + (100. * u.arcsec)
```

```
bkg_ap = SkyCircularAnnulus(position, r_in=r_in, r_out=r_out)
bkg_ap_pix = bkg_ap.to_pixel(wcs)
bkg_ap_pix.plot(color='red', lw=3, ls='--', alpha=0.9)


# MIRI Imager
r = self.radius * u.arcsec
r_in = r  - (37. * u.arcsec)
r_out = r  + (37. * u.arcsec)
arc_ap = SkyCircularAnnulus(position, r_in=r_in, r_out=r_out)
arc_ap_pix = arc_ap.to_pixel(wcs)
arc_ap_pix.plot(color='cyan', lw=3, ls=':', alpha=0.9)

axs.set_facecolor('black')
axs.coords.grid(True, color='white', ls='dotted')
axs.coords[0].set_axislabel('Right Ascension (J2000)')
axs.coords[1].set_axislabel('Declination (J2000)')

# indicative MIRI FOV
x = Angle(self.radius, u.arcsec)
y = x.degree
t3_pos = SkyCoord((self.ra_deg) * u.degree, (self.dec_deg + y)
ap3 = SkyRectangularAperture(t3_pos, 74 * u.arcsec, 113 * u.arcs
ap3_pix = ap3.to_pixel(wcs)
ap3_pix.plot(color='yellow', lw=3, ls='-', alpha=0.9)

axs.set_facecolor('black')
axs.coords.grid(True, color='white', ls='dotted')
axs.coords[0].set_axislabel('Right Ascension (J2000)')
axs.coords[1].set_axislabel('Declination (J2000)')

# display the plot
plt.tight_layout()

# save as a pdf
plot_name = os.path.join(str(self.name), str(self.name) + '_' +
                         str(self.wavelength) + '.pdf')
try:
    os.remove(plot_name)
```

```
        except:
            pass

        fig.savefig(plot_name, dpi=200)

    def show_image(self, MCSNR, test_src_coord, wavelength):
        """
        show_image plots cut images for the user to see when running th
        """
        my_image_file = os.path.join(str(self.name), str(self.sensor) +

        # load the file data, header, and wcs
        with fits.open(my_image_file) as hdulist:
            header = hdulist[0].header
            data = hdulist[0].data
            wcs = WCS(header)
            size = self.radius * 4
            arc_pix = header['PXSCAL1']
            if arc_pix < 0:
                arc_pix = -arc_pix
            size_pix = size / arc_pix
            factor = 1
            if size_pix in range(100,200):
                factor = (2160/162)
            if size_pix in range(300,400):
                factor = (2160/324)
            if size_pix in range(500,600):
                factor = (2160/529)

            # changing to two twice the SNR diameter
            im_size = (self.radius * 4) / arc_pix
            im_size = int(im_size)

            position = SkyCoord(test_src_coord[0] * u.degree, test_src_c
            cutout = Cutout2D(data, position, (im_size), wcs=wcs)

            # save the cutout to fits
            fits_name = os.path.join(str(self.name), str(self.name) + '
                                     str(self.wavelength) + '_cut.fits '
```

```
            new_hdu = fits.PrimaryHDU(data=cutout.data)
            new_hdulist = fits.HDUList([new_hdu])
            new_hdulist.writeto(fits_name, overwrite=True)

            # fill the point sources
            cutout.data = self.fill_sources(cutout.data, save_plot=True

            # save the filled image to fits
            fits_name = os.path.join(str(self.name), str(self.name) + '
                                     str(self.wavelength) + '_cut_ps-rem
            new_hdu = fits.PrimaryHDU(data=cutout.data)
            new_hdulist = fits.HDUList([new_hdu])
            new_hdulist.writeto(fits_name, overwrite=True)

            self.plot_image(cutout.data, cutout.wcs, coords=test_src_coo

        return my_image_file

    def ap_phot(self, my_image_file, Rad, test_src_coord, wavelength):
        """
        ap_phot applies aperture photometry to calculate the flux of th
        the source referencing the radius of the snr as the radius of t
        apertures away from the source which are then averaged and subt
        Input
        =========================
        my_image_files (.fits):
            .fits file from coordinates
        Rad (float):
            radius in units of arcsec
        test_src_coord (float):
            coordinates of target in decimal degrees
        wavelength (float):
            waveband in units of um
        Output
        =========================
        FluxIR (float):
            calculated flux in units of erg/s/cm^2
        """
```

```
fluxes = []

r = Rad * u.arcsec
r_in = r + (40. * u.arcsec)
r_out = r + (100. * u.arcsec)

# load the file data, header, and wcs
with fits.open(my_image_file) as hdulist:
    my_hdu = hdulist[0]
    my_hdu.data = np.nan_to_num(my_hdu.data)
    pixel_area = my_hdu.header['PXSCAL1']**2

    print('Running aperture photometry {}-{} um'.format(self.nam
    position = SkyCoord(test_src_coord[0] * u.degree, test_src_c
    apertures = SkyCircularAperture(position, r=Rad * u.arcsec)
    phot_table = aperture_photometry(my_hdu, apertures)
    fluxes.append(phot_table['aperture_sum'])
    #print(phot_table['aperture_sum'])


    print('Running background aperture photometry {}-{} um'.form

    bkg_ap = SkyCircularAnnulus(position, r_in=r_in, r_out=r_ou
    phot_table_bkg = aperture_photometry(my_hdu, bkg_ap)
    #print(phot_table_bkg['aperture_sum'])

    # bkg subtract
    flux_bkg_sub = phot_table['aperture_sum'] - phot_table_bkg[

# convert to Jy
if flux_bkg_sub.value > 0:
    ujy_arcsec = flux_bkg_sub.value * 23.5045
    Jy = ujy_arcsec * pixel_area * 1.e-06
    Jy = Jy[0]

    erg = (Jy * 1.e-23) * (2.997924e14 / ((wavelength)**2))

    self.flux_Jy = Jy
    self.flux_erg = erg
```

```
        else:
            erg = 0.0

            self.flux_Jy = 0.0
            self.flux_erg = 0.0

        return erg

    def flux_arc(self, my_image_file, Rad, test_src_coord, wavelength):

        r = Rad * u.arcsec
        arc_r_in = r - (37. * u.arcsec)
        arc_r_out = r + (37. * u.arcsec)
        bkg_r_in = r * 1.1
        bkg_r_out = r * 1.25

        # load the file data, header, and wcs
        with fits.open(my_image_file) as hdulist:
            my_hdu = hdulist[0]
            my_hdu.data = np.nan_to_num(my_hdu.data)
            pixel_area = my_hdu.header['PXSCAL1'] ** 2

            position = SkyCoord(test_src_coord[0] * u.degree, test_src_c

            print('Running arc aperture photometry {}-{} um'.format(sel
            arc_ap = SkyCircularAnnulus(position, r_in=arc_r_in, r_out=
            phot_table_arc = aperture_photometry(my_hdu, arc_ap)

            bkg_ap = SkyCircularAnnulus(position, r_in=bkg_r_in, r_out=
            phot_table_bkg = aperture_photometry(my_hdu, bkg_ap)

            # bkg subtract
            flux_bkg_sub = phot_table_arc['aperture_sum'] - phot_table_b

        # convert to Jy
        if flux_bkg_sub.value > 0:

            ujy_arcsec = flux_bkg_sub.value * 23.5045
```

```
            Jy = ujy_arcsec * pixel_area * 1.e-06
            Jy = Jy[0]

            erg = (Jy * 1.e-23) * (2.997924e14 / ((wavelength)**2))

            # we must divide the circumference by the MIRI imager FOV h
            # average flux in a MIRI Imager FOV
            circum = 2 * np.pi * Rad
            n_miri_ima = circum / 113.

            self.arc_flux_Jy = Jy / n_miri_ima
            self.arc_flux_erg = erg / n_miri_ima

        else:
            self.arc_flux_Jy = 0.0
            self.arc_flux_erg = 0.0

        return self.arc_flux_erg


    def run(self):
        """
        run the class methods
        """
        # Eq2Deg Function
        test_src_coord = self.eq2deg(self.ra, self.dec)

        # Query Function
        self.query(test_src_coord, self.sensor, self.wavelength)

        # Cuts Function
        self.cuts(self.wavelength, self.sensor, self.name)

        # Plot Image function
        my_image_file = self.show_image(self.name, test_src_coord, self

        # Ap Phot Function
        flux = self.ap_phot(my_image_file, self.radius, test_src_coord,
```

```
flux_arc = self.flux_arc(my_image_file, self.radius, test_src_co

return self.flux_Jy, self.flux_erg, self.arc_flux_Jy, self.arc_
```

Bibliography

[1] Destruction of Interstellar Dust in Evolving Supernova Remnant Shock Waves. *Slavin et al,* The Astrophysical Journal, 10 Apr 2015.

[2] Infrared Emission from Supernova Remnants: Formation and Destruction of Dust. *Brian J. Williams et al,* The Astrophysical Journal, 3 Nov 2017.

[3] The population of X-ray supernova remnants in the Large Magellanic Cloud. *P. Maggi et al,* Astronomy Astrophysics, 20 Oct 2015.

[4] Spitzer Survey of the Large Magellanic Cloud: Surveying the Agents of a Galaxy's Evolution (SAGE). *Meixner et al,* The Astronomical Journal, Dec 2006.

[5] A Survey of Infrared Supernova Remnants in the Large Magellanic Cloud. *Seok,* The Astrophysical Journal, 20 Dec 2013.

[6] New SNRs in the LMC (not yet published). *P. J. Kavanagh et al.*

[7] VizieR Catalog Website.
`http://vizier.u-strasbg.fr/viz-bin/VizieR`

[8] Spitzer ST Documentation
`http://www.spitzer.caltech.edu/`

[9] James Webb ST Documentation
`https://jwst-docs.stsci.edu/display/HOM/JWST+User+Documentation+Home`

[10] JWST Exposure Time Calculator
https://jwst.etc.stsci.edu/

[11] MIRISim Website
http://miri.ster.kuleuven.be/bin/view/Public/MIRISim$_{P}ublic$

All websites last accessed 10:00, 30th Aug 2018.