The Kilonova Data Challenge

Arman A. Svoboda, Undergraduate

ABSTRACT

Kilonovae are rare transient astronomical phenomena where two neutron stars, or a neutron star and a black hole collide. As a result, a massive amount of energy and light is released. We are conducting our research in anticipation of the Rubin Observatory, and its ten-year Legacy Survey of Space and Time (LSST). In preparation for this, we are researching how to find kilonovae in an image. Our project uses simulated images from a data set built for the LSST Dark Energy Science Collaboration (LSST-DESC), called the second data challenge (DC2). DC2 is a simulated image set containing galaxies, stars, supernovae, and variable stars. However, these simulated images do not contain kilonovae, so we must add them using synthetic source injection. We are taking existing images and adding artificial objects to them. Thus far, we have built several critical Jupyter notebooks. These notebooks are hosted in the Rubin Science Platform (RSP), which allows us to access and analyze DC2 data and LSST functions. One of these notebooks displays a set of images in a mosaic, allowing us to see the full area into which a kilonovae will be added. This notebook visualizes a larger area of the focal plane of the camera. The other notebook we developed displays DC2 images that highlight the host galaxies we chose for a kilonova. These notebooks we developed will be important tools in bringing the Kilonova Data Challenge to a conclusion.

I. INTRODUCTION

Dark energy, and more specifically transients, are responsible for the ever-expanding rate of the universe. As a result, we have begun studying these topics extensively. Organizations such as the Legacy Survey of Space and Time Dark Energy Science Collaboration (LSST-DESC), and the Laser Interferometer Gravitational Wave Observatory (LIGO) have formed to gain a better understanding of this phenomenon. LSST is a ten-year survey that will be conducted at the Rubin Observatory in Chile. The Rubin Observatory features the largest digital camera ever made. It will cover approximately 20,000 square degrees of the sky, and amass over 250 petabytes worth of data after concluding its ten-year survey. In only a year alone, LSST will have gathered more data than every other telescope combined. And as a result, astronomers from across the world will be searching for numerous objects within that data. More specifically for us, we will be searching for kilonova. Meanwhile, LIGO is using a gravitational wave detector to detect gravitational waves from the distant universe.

Kilonovae fall under the classification of a transient. Transients are astronomical phenomena that are partially or completely destroyed and are extremely short-lived events. Kilonvoa specifically are when two neutron stars or a neutron star and a black hole collide. This results in a massive amount of energy and light being released. They also release an abundance of heavy elements such as gold, platinum, and uranium. Thus far, only a single kilonova was found, GW170817. This was a gravitational wave found by LIGO and was the first time a ripple in space-time was detected. The reason we want to search for kilonova is because transients are a source of heavy elements, which can create objects just like Earth. But also because we can study their gravitational waves and redshifts. A redshift is an electromagnetic wave that will get longer because the universe is expanding. Therefore we can measure the expansion rate of the universe using redshifts and gravitational waves.

Our goal in this research is to determine where kilonova are in the sky using a simulated image set built by The Dark Energy Science Collaboration (DESC) called the second data challenge (DC2). We are using these images as a base to add artificial kilonovae into them. This will allow us to mimic actual LSST data and search for kilonovae. However, these images only contain galaxies, stars, supernovae, and variable stars. So we are using synthetic source injection to inject kilonova into a simulated image set. And then we are developing code that will detect and locate where the kilonova we injected is in the image. This will allow us to test our ability to search for kilonova in an image. With this, we will be able to exercise how to find kilonovae in preparation for when actual Rubin data is taken.

II. PROGRESS

1. The Rubin Science Platform

The Rubin Science Platform (RSP) is an online platform for scientists to access Rubin LSST data and develop Jupyter notebooks using this data. We are using this platform as a means to develop notebooks critical to our goal of finding kilonova. The RSP acts as a repository for everything LSST data related. LSST-specific functions and pipelines such as the butler, lsst.afw.display, and image coaddition, etc which we use in this project. The images we're working with are all extremely large, with every single pixel being a form of data. So the RSP allows us to use these images without having to download them. It streamlines our entire development process by having everything available on one platform.

2: Visualizing One Square Degree of The Sky

Our first goal was to develop a program to visualize a larger area of the focal plane of the camera. We wanted to do this so that we could see the full area the Kilonova Data Challenge (KDC) would cover. To accomplish this we developed the Mosaic Viewer. We used the RSP to host and develop this notebook. Since the Rubin Observatory is still under construction, the RSP contains simulated DC2 data. This simulated image set contains galaxies, stars, supernovae, and variable stars. However, since there is no kilonovae in any of the images, we have to add them to the images. For the time being, we created point sources in the image to highlight where kilonovae will be.

As it was mentioned previously, we needed to display a larger area of the focal plane. To accomplish this we developed the Mosaic Viewer notebook. To begin, we first developed code to print 189 images, which is the equivalent of the 189 charge-coupled devices (CCDs) LSST has. This step involved a lot of problem-solving as the kernel would crash due to using too much memory when trying to successfully run all 189 images. So we created a remove image function, and we were then able to successfully run all 189 images. After successfully printing all of the images in sequential order, we began developing the rest of the notebook. Now we were working to display the images in a mosaic.

Built into the RSP is a firefly function, which visualizes Flexible Image Transport System (FITS) images. Using the firefly function, the mosaic will display in another window allowing us to see the 1 square degree of the sky. We began by importing the functions that will display a set of calibrated exposure images. Which can be seen in the first cell of the code (Figure 3 for code). Next, we had to call the butler to grab the DC2 images from the DP0.2 catalog. The butler functions similar to that of an actual butler. We can call the butler and it will retrieve what we ask it to. In this case, we are calling the butler to load in the DP0.2 tables. With the DP0.2 tables now loaded, we define how many images we want to display, and what chip we want to start the display on. This is done by setting a variable "x" equal to six in this case. Then we define what

chip to start on, this can be anywhere from 1 to 189. Lastly in this cell, we have to define what display function we want the images to display in. They could be displayed in either matplotlib or Firefly, but we're going to be using Firefly. In the last cell, we use the butler to call calibrated exposure (calexp) images for the set number of different detectors. The cell begins by defining the size of the figure. Then the mosaic command is used from an LSST pipeline (lsst.afw.display) to set the background, mode, and gutter. Next, we used a for loop to run through the first detector that we set a value for, in Figure 3 it's set to twenty-one. And then it uses the butler to call the image to the x value we set. There were a lot of initial issues getting the loop to work for all 21 images. Initially, only detectors five through eight, and twelve would only display. To try and understand what was happening, we hardcoded each image value into the code. An example of how the images were defined is im1 = butler.get('calexp', **dataId). With this all twenty-one images would display, so it was concluded there was an error with the for loop, and not the images. After revisiting the loop and debugging it line by line, the issue was the way the butler was called. It wasn't calling all of the images. After fixing this, all of the twenty-one images were called without issue. To finally display them we created an if statement to choose if the image will be displayed in Firefly or matplotlib. Either of the two functions can be commented out for the preference you have when displaying these images. Lastly, the display and scale are written out to display the image and draw its labels.

This notebook achieved what we wanted it to after development throughout the summer. All twenty-one images are displayed in a mosaic, showing us one square degree of the sky (Figure 1). Which in turn lets us view the entire focal plane the Kilonova Data Challenge covers.

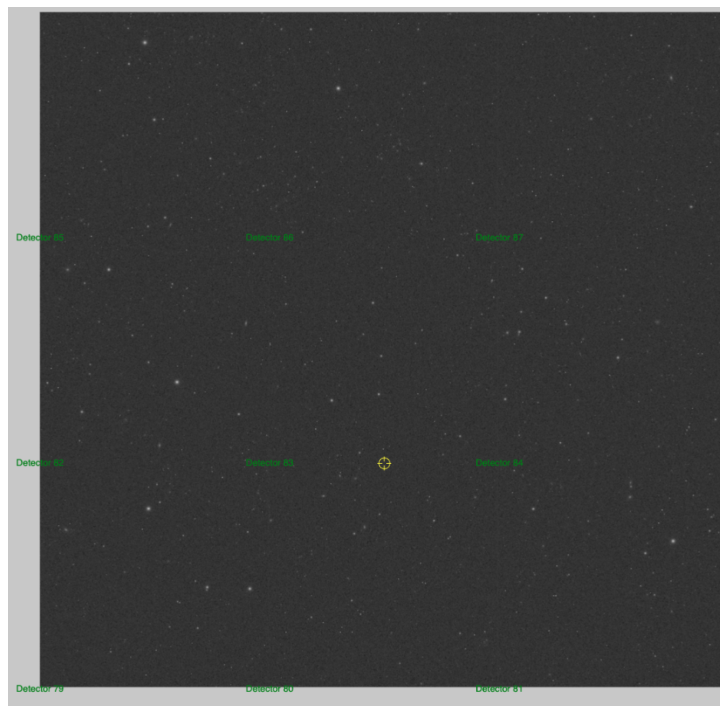

Figure 1. The Mosaic Viewer notebook output of 1 square degree of the sky displayed.

3: Displaying Images in DP0.2

The Data Image Preview 0.2 (DP0.2) simulated image set was created in anticipation of the start of Rubin operations, Using DP0.2 data, we wanted to develop code to display an image and highlight where the galaxies are where we chose to host kilonova. To accomplish this we created a Jupyter notebook called Galaxy Viewer. We created this notebook at first to just visualize images for all 20 locations. This was done by importing the basic functions of the notebook, such as numpy, pandas, the butler, and more LSST-specific functions. This can be seen in Figure 4. Then we defined the right ascension and declination along with the tract and patch. This gives us a tract and patch for the coadd associated with a particular point. Lastly, we defined the full display of the image and the cutout coadd. This would print an image highlighting the galaxies where kilonovae were chosen to be hosted. The next step was to view a single calexp image after we used synthetic source injection to find a visit and band. First, we needed to create a loop to print all of the 20 images sequentially. Initially, we ran into errors with actually calling the butler to get the images. After ironing that out, we were able to view several images sequentially. And in the following cell, we can just view one of these images.

Furthermore, we also developed how to view a coadd at a particular location. We find the tract and patch to get the dataid for that specific coordinate, and then we use the full_image function to obtain and visualize the coadd. This is highlighted with a green circle. And the red circle is just the central coordinate. After this, we call the cutout image to get a cutout of that image at that specific location. This results in an image being displayed, which highlights a galaxy that we are choosing to host a kilonova (Figure 2).

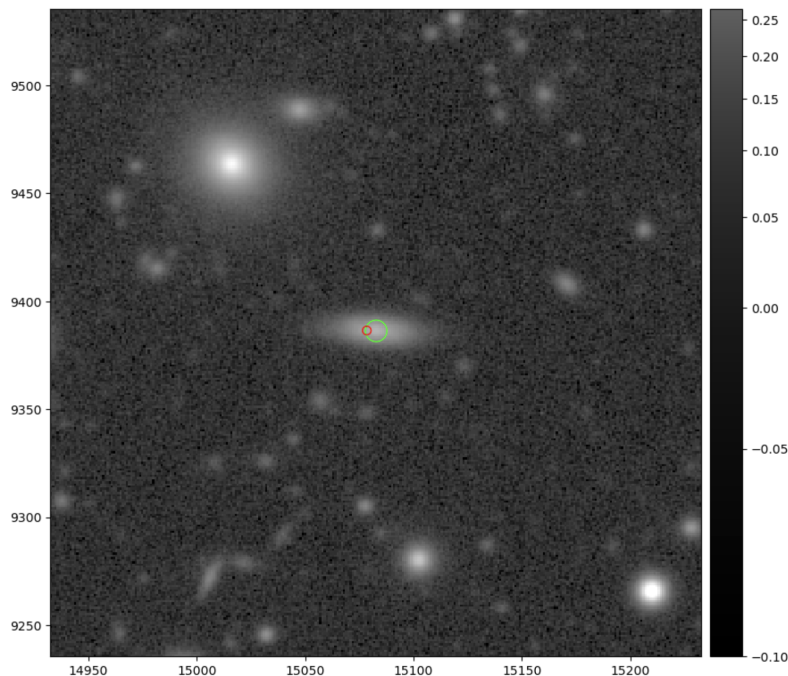

Figure 2. This is the last output of the Galaxy Viewer notebook. It shows the highlighted galaxy we chose to host a kilonova.

III. CONCLUSIONS

This summer we progressed extremely far in our project. We've developed, and are still developing critical notebooks that will assist in bringing us closer to a conclusion of determining how well kilonova will be found in LSST data. The Mosaic viewer notebook allowed us to view the entire focal plan the KDC will cover. Showing us one square degree of the sky. Meanwhile, the Galaxy Viewer notebook shows us the galaxies we are choosing to host kilonova. In addition to this, we will be developing the process of displaying supernova light curves, as well as injecting artificial kilonova into DC2 images. We ran into a lot of technical issues throughout this project thus far. However, through problem-solving, we were able to overcome a lot of these issues. The next steps going forward will be to answer our question of how well we can find kilonovae in LSST data. We will base our conclusions on how well we find kilonova in the simulated images. As of now, we are predicting only two will be visible. We also need to determine how we can tell kilonova apart from supernovae. There are still several questions to answer, but this summer has brought us one step closer to concluding this project.

IV. ACKNOWLEDGMENTS

This manuscript has been authored by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics. I would also like to especially thank Dr. Matthew Wiesner and Dr. Douglas Tucker for many hours of guidance.

This work was supported in part by the U.S. Department of Energy, Office of Science, Office of Workforce Development for Teachers and Scientists (WDTS) under the Visiting Faculty Program (VFP).

V. REFERENCES

[1] Abolfathi, B., Alonson, D., Armstrong, R.,…Wiesner, M., et al 2021, "The LSST DESC DC2 Simulated Sky Survey." Astrophysical Journal Letters, 2021.
[2] Kasen, D., Metzger, B., Barnes, J. et al 2017. "Origin of the heavy elements in binary neutron-star mergers from a gravitational-wave event," Nature 551, 80–84 (2017). https://doi.org/10.1038/nature24453. Code for kilonova models available here: https://github.com/dnkasen/Kasen_Kilonova_Models_2017
[3] Soares-Santos, M., Holz, D. E., Annis, J., et al. 2017, "The Electromagnetic Counterpart of the Binary Neutron Star Merger LIGO/Virgo GW170817. I. Dark Energy Camera Discovery of the Optical Counterpart," The Astrophysical Journal, 848, L16, 981, arXiv:1710.05459.
[4] Codebase for this project available at: https://github.com/mpwiesner/KDC

Figure 3. The Mosaic Viewer Notebook Code

Displays a set of calibrated exposures in a mosaic, so we can view 1 square degree

```
import warnings
import matplotlib.pyplot as plt

from lsst.afw.display import Mosaic
from lsst.daf.butler import Butler
import lsst.afw.image as afwImage
import lsst.afw.display as afwDisplay
afwDisplay.setDefaultBackend('matplotlib')
import gc
```

Last executed at 2023-07-25 16:29:33 in 2.30s

Call the Butler for DP0.2

```
#Call the Butler, load in DP0.2 tables

butler = Butler('dp02', collections='2.2i/runs/DP0.2')
registry = butler.registry
```

Last executed at 2023-07-25 16:29:49 in 3.05s

How many images do you want to display? What chip do you want ot start the display on?

```
x = 6  #This is the number of chips you would like to display
chip_start = 79  #This is the chip you want to start on (from 1 to 189)
display_in = 'firefly' #Either 'firefly' or 'matplotlib'
```

Last executed at 2023-07-25 16:29:50 in 2ms

Use the Butler to call calexps for x number of different detectors.

```
#We first choose the size of the figure.
fig = plt.figure(figsize=(20,20))

#This uses the mosaic command from lsst.afw.display.mosaic: https://pipelines.lsst.io/py-api/lsst.afw.display.Mosaic.html
m = Mosaic()
m.setGutter(5)
m.setBackground(10)
m.setMode("square")

labels=[]  #This makes sure the labels and images lists start empty
images= []

#This loop goes from the first detector above (chip_start) to the xth image and uses butler.get to obtain the image

for i in range(0,x):
    name = "im"+str(i) #This is used to define the image variable name
    namer = "im"+str(i)+".image"  #This is used to populate the list of image names
    labels.append("Detector "+str(chip_start+i)) #this produces the list of image labels that will be displayed
    datad = "dataId"+str(i)
    datad = {'visit': 1185205, 'detector': chip_start + i, 'band': 'r'} #This gives the visit, detector number and band for the Butler
    vars()[name] = butler.get('calexp', **datad)  #This does the actual call to the Butler and puts the data in an array called im1, im2, etc.
    images.append(namer)

images2 = [eval(string) for string in images] #This converts the list of strings to a list of string literals

mosaic = m.makeMosaic(images2) #This actually creates the mosaic

#This if statement determines if the mosaic is displayed in firefly or matplotlib

if display_in == 'firefly':
    afwDisplay.setDefaultBackend('firefly') #This opens it in firefly. Uncomment below if you want it in matplotlib.
    display = afwDisplay.Display(frame=1)

else:
    afwDisplay.setDefaultBackend('matplotlib')
    display = afwDisplay.Display(frame=fig)

#This sets the display scale, displays the image and draws the labels
display.scale('asinh', 'zscale')
display.mtv(mosaic)
m.drawLabels(labels, display)
```

Figure 4. The Galaxy Viewer Notebook Code

```python
import numpy as np
import matplotlib
import pandas as pd
import gc
# allow for matplotlib to create inline plots in our notebook
%matplotlib inline
import matplotlib.pyplot as plt      # imports matplotlib.pyplot as plt

import lsst.daf.butler as dafButler     # gen3 butler
from astropy.wcs import WCS          # imports astropy's World Coordinate System function WCS

import lsst.sphgeom
import lsst.afw.display as afwDisplay
import lsst.afw.image as afwImage
import lsst.afw.display.utils as dispUtils
import lsst.geom as geom
from lsst.geom import SpherePoint, Angle
afwDisplay.setDefaultBackend('matplotlib')


butler = dafButler.Butler('dp02', collections='2.2i/runs/DP0.2')
filter_id = 'r'
```

Last executed at 2023-07-26 13:37:56 in 4.81s

```python
#Read KN host galaxy positions and KN offsets from the csv file
#a = pd.read_csv('input.txt', delim_whitespace = True)
#ra1 = a['RA']
#dec1 = a['DEC']
#print(len(ra1))

galaxies = pd.read_csv('truth_DP0_phosim_real.csv')
ra1 = galaxies['ra_1']
dec1 = galaxies['dec_1']
ra_off = galaxies['RA_offset']
dec_off = galaxies['DEC_offset']
kn_ra1 = ra1+ra_off
kn_dec1 = dec1+dec_off
```

Last executed at 2023-07-26 13:40:51 in 478ms

•••

```python
def remove_figure(fig):
    """
    Remove a figure to reduce memory footprint.

    Parameters
    ----------
    fig: matplotlib.figure.Figure
        Figure to be removed.

    Returns
    -------
    None
    """
    # get the axes and clear their images
    for ax in fig.get_axes():
        for im in ax.get_images():
            im.remove()
    fig.clf()       # clear the figure
    plt.close(fig)  # close the figure
    gc.collect()    # call the garbage collector
```

Last executed at 2023-07-25 10:29:17 in 4ms

```python
# Find the tract and patch for the coadd associated with a particular point
def find_tract_patch(ra, dec):
    my_spherePoint = lsst.geom.SpherePoint(ra*lsst.geom.degrees, dec*lsst.geom.degrees)
    skymap = butler.get('skyMap')
    my_tract = skymap.findTract(my_spherePoint)
    my_patch = my_tract.findPatch(my_spherePoint)
    my_patch_id = my_tract.getSequentialPatchIndex(my_patch)
    tract1 = str(my_tract)
    patch1_ok = str(my_patch_id)
    tract1_ok = tract1[13:17]
    tract = int(tract1_ok)
    patch = int(patch1_ok)
    dataId = {'band': filter_id, 'tract': int(tract1_ok), 'patch':int(patch1_ok)}
    return dataId
```

```python
#Shows the full image view
def full_image(ra, dec, dataId):
    datasetType = 'deepCoadd'
    coadd = butler.get(datasetType, **dataId)
    radec = SpherePoint(ra, dec, lsst.geom.degrees)
    xy = coadd.getWcs().skyToPixel(radec)
#    fig, ax = plt.subplots(xdim, ydim, figsize=(10, 10))
#    plt.sca(ax[0])

    fig = plt.figure(figsize=(20,20))
    display = afwDisplay.Display(frame=1, backend='matplotlib')
    display.scale("asinh", "zscale")
 #   display.mtv(image.getMaskedImage().getImage())

 #   coadd.image.writeFits('output.fits')
    display.mtv(coadd.image)
    display.dot('o', xy.getX(), xy.getY(), ctype='green', size = 20)
  #  display.dot('o', xy.getX(), xy.getY(), ctype='green', size =40)
 #   plt.show()
  #  display.mtv(image.getMaskedImage().getImage())
    return
```

Last executed at 2023-07-25 10:29:19 in 4ms

```python
def cutout_coadd(butler, ra, dec, dataid, band='r', datasetType='deepCoadd',
                 skymap=None, cutoutSideLength=100, **kwargs):
    """
    Produce a cutout from a coadd at the given ra, dec position.

    Adapted from DC2 tutorial notebook by Michael Wood-Vasey.

    Parameters
    ----------
    butler: lsst.daf.persistence.Butler
        Helper object providing access to a data repository
    ra: float
        Right ascension of the center of the cutout, in degrees
    dec: float
        Declination of the center of the cutout, in degrees
    band: string
        Filter of the image to load
    datasetType: string ['deepCoadd']
        Which type of coadd to load.  Doesn't support 'calexp'
    skymap: lsst.afw.skyMap.SkyMap [optional]
        Pass in to avoid the Butler read.  Useful if you have lots of them.
    cutoutSideLength: float [optional]
        Size of the cutout region in pixels.

    Returns
    -------
    MaskedImage
    """
    radec = geom.SpherePoint(ra, dec, geom.degrees)
    cutoutSize = geom.ExtentI(cutoutSideLength, cutoutSideLength)

    if skymap is None:
        skymap = butler.get("skyMap")

    # Look up the tract, patch for the RA, Dec
    tractInfo = skymap.findTract(radec)
    patchInfo = tractInfo.findPatch(radec)
    xy = geom.PointI(tractInfo.getWcs().skyToPixel(radec))
    bbox = geom.BoxI(xy - cutoutSize // 2, cutoutSize)
    patch = tractInfo.getSequentialPatchIndex(patchInfo)

#  coaddId = {'tract': tractInfo.getId(), 'patch': patch, 'band': band}
    parameters = {'bbox': bbox}

    cutout_image = butler.get(datasetType, parameters=parameters,
                              dataId=dataid)

    return cutout_image
```

This is a routine to view single calexps, after we used Synthetic Source to find visit and band

Here we used a loop to view several of them in a row.

```
for i in range(1,4):
    fig, ax = plt.subplots(figsize=(20,20))
    dataId1 = {'visit': 941094, 'detector': int(i), 'band': 'i'}
    im1 = butler.get('calexp', **dataId1)
    display = afwDisplay.Display(frame=fig)
    display.scale('asinh', 'zscale')
    display.mtv(im1.image)
```

Last executed at 2023-07-26 13:41:10 in 12.33s

```
#Here we view just one

dataId1 = {'visit': 941094, 'detector': 2, 'band': 'i'} #6,7,15,16
im1 = butler.get('calexp', **dataId1)
#dataId2 = {'visit': 941094, 'detector': 3, 'band': 'i'} #6,7,15,16
#im2 = butler.get('calexp', **dataId2)
#dataId3 = {'visit': 941094, 'detector': 4, 'band': 'i'} #6,7,15,16
#im3 = butler.get('calexp', **dataId3)
#dataId4 = {'visit': 941094, 'detector': 5, 'band': 'i'} #6,7,15,16
#im4 = butler.get('calexp', **dataId4)
#dataId5 = {'visit': 941094, 'detector': 6, 'band': 'i'} #6,7,15,16
#im5 = butler.get('calexp', **dataId5)

ra1 = 59.7288694,
dec1 = -36.6912823
ra2 = 59.5252768
dec2 = -36.6754221

fig = plt.figure(figsize=(20,20))
display = afwDisplay.Display(frame=fig)
display.scale('asinh', 'zscale')
display.mtv(im1.image)

#radec1 = SpherePoint(ra1, dec1, lsst.geom.degrees)
#xy = coadd.getWcs().skyToPixel(radec1)
#display.dot('o', xy.getX(), xy.getY(), ctype='green', size = 20)

#radec2 = SpherePoint(ra2, dec2, lsst.geom.degrees)
#xy = coadd.getWcs().skyToPixel(radec2)
#display.dot('o', xy2.getX(), xy2.getY(), ctype='green', size = 20)


plt.show()
remove_figure(fig)
```

Here we view a coadd at a particular location.

We use find_tract_patch to get the dataid for a particular coordinate and then we use the full_image function to get and display the coadd.

Note that we mark the input ra and dec with a green circle.

```
#ra = 57.6491576 edge on galaxy
#dec = -39.26560194
#ra = 59.6491576
#dec = -37.26560194
#dataid = find_tract_patch(ra,dec)
#full_image(ra,dec,dataid)



i=15
#for i in range(1,2):
ra = 59.65688129
dec = -36.75083078
dataid = find_tract_patch(ra,dec)

full_image(ra,dec,dataid)

#    input()
```

Last executed at 2023-07-26 13:41:31 in 13ms

Here we call cutout_image to get an image cutout at a particular location

Then we show the location of the galaxy with a green circle and of the kilonova with a red circle.

```
#for i in range(0,len(ra1)):
ra = ra1[0]
dec = dec1[0]
kn_ra = kn_ra1[0]
kn_dec = kn_dec1[0]

#    kn_ra = kn_ra1[i]
#    kn_dec = kn_dec1[i]
dataid = find_tract_patch(ra,dec)
cutout_image = cutout_coadd(butler, ra, dec, dataid, band='i', datasetType='deepCoadd',cutoutSideLength=300
                           )
   # fig = plt.figure(figsize=(40,40))
fig, ax = plt.subplots(figsize=(10,10))
radec = SpherePoint(ra, dec, lsst.geom.degrees)
xy = cutout_image.getWcs().skyToPixel(radec)

radec2 = SpherePoint(kn_ra, kn_dec, lsst.geom.degrees)
xy2 = cutout_image.getWcs().skyToPixel(radec2)

display = afwDisplay.Display(frame=fig)
display.scale('asinh', 'zscale')
display.mtv(cutout_image.image)

display.dot('o', xy.getX(), xy.getY(), ctype='green', size = 5)
display.dot('o', xy2.getX(), xy2.getY(), ctype='red', size = 2)

#    plt.title('Object '+str(i)+' at RA = '+str(ra)+' and dec = '+str(dec))
#    plt.show()
#    remove_figure(fig)
```