LAB 3 CODE

December 13, 2020

0.1 Antony Sikorski - PHYS 164 - Fall 2020

0.1.1 Lab 3 Code

0.2 Loading in FITS files

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from astropy.io import fits
import os
from numpy.linalg import inv
```

[2]:
```
data_dir = './data_folder/'
data_files = np.array(sorted(os.listdir(data_dir)))
```

[3]:
```
def load_headers_all_files(headers):

    #Reads and returns multiple FITS headers from all files.

    output = np.full([len(headers), len(data_files)], "", dtype = object)

    # Reads the headers from all files

    i = 0
    j = 0

    for i, data_file in enumerate(data_files):
        h = fits.open(data_dir + data_file)
        for j, header_name in enumerate(headers):
            output[j, i] = h[0].header[header_name]
    h.close()

    return output
```

[4]:
```
import pandas
headers = ['OBSTYPE', 'OBJECT', 'EXPTIME', 'FILTNAM', 'COVER']
obstypes, objects, exptimes, filters, overscans =␣
 ↪load_headers_all_files(headers) # Calling the defined function here
headers_lists = list(zip(obstypes, objects, exptimes, filters, overscans))
```

```
fits_dataframe1=pandas.DataFrame(headers_lists, columns=headers)
fits_dataframe1
```

[4]:

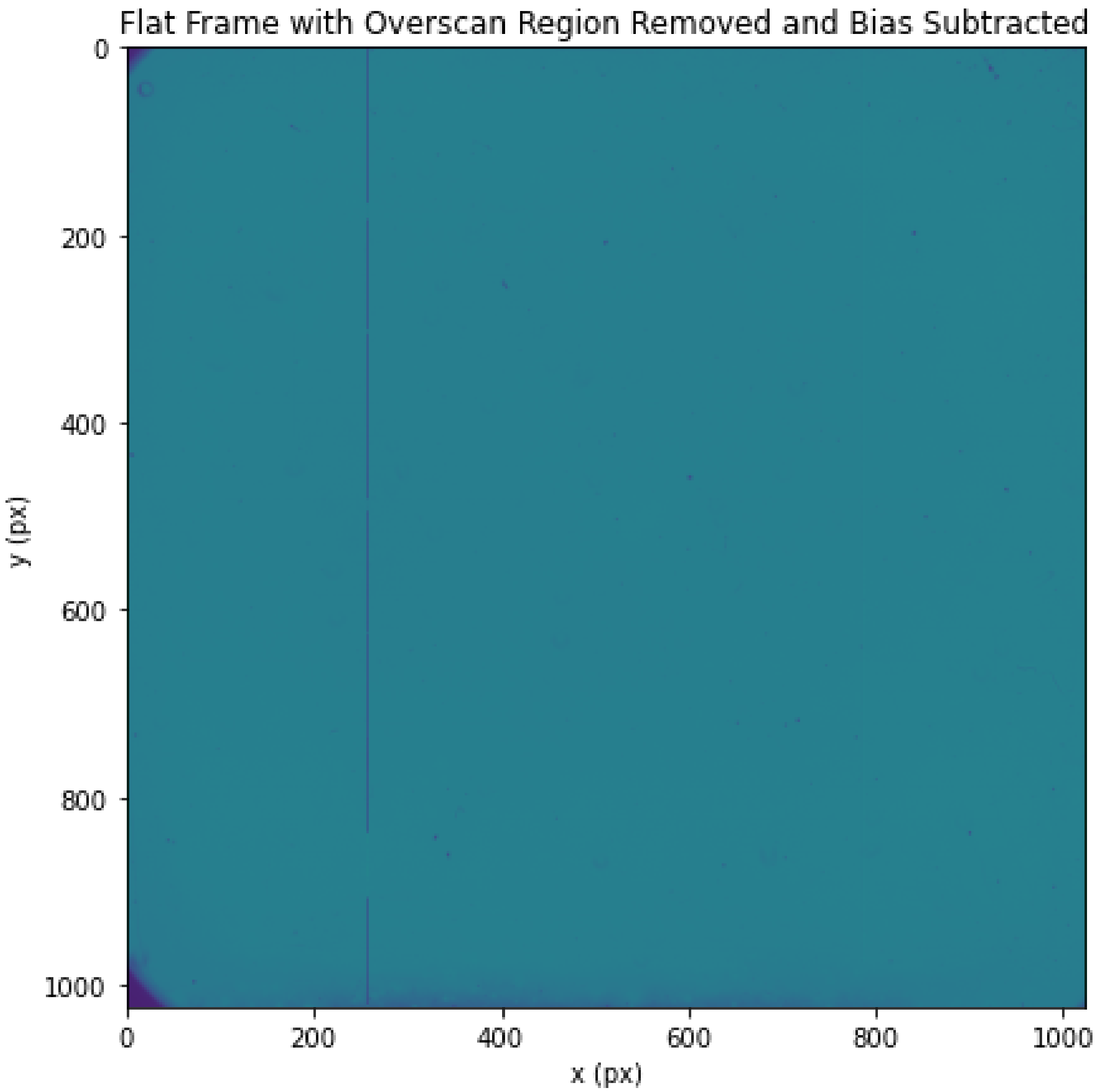
	OBSTYPE	OBJECT	EXPTIME	FILTNAM	COVER
0	OBJECT	bias	0	R	32
1	OBJECT	flatfield	3	R	32
2	OBJECT	Parthenope - Group A	30	R	32
3	OBJECT	flatfield_dark	3	R	32
4	OBJECT	Parthenope	30	R	32

0.3 Data Reduction

[5]:
```
first_flat = data_files[objects == 'flatfield'][0]

def load_frame_overscan_remove_bias_subtract(filename):

    # simply modified reduction function from the last lab
    flat = fits.getdata(data_dir+filename)
    bias = fits.getdata(data_dir+data_files[0])*1.000000001
    flat=flat-bias
    image = flat[:,:1024]

    return image
```

[6]:
```
def plot_fits_im(ax, data, xlabel='x (px)', ylabel='y (px)', title='',␣
 ↪**imshow_kwargs):
    ax.imshow(data, **imshow_kwargs)
    ax.set_xlabel(xlabel)
    ax.set_ylabel(ylabel)
    ax.set_title(title)
```

[7]:
```
data = load_frame_overscan_remove_bias_subtract(first_flat)
fig1, ax1 = plt.subplots(figsize = [7, 7])
plot_fits_im(ax1, data, title='Flat Frame with Overscan Region Removed and Bias␣
 ↪Subtracted')
fig1.show()
```



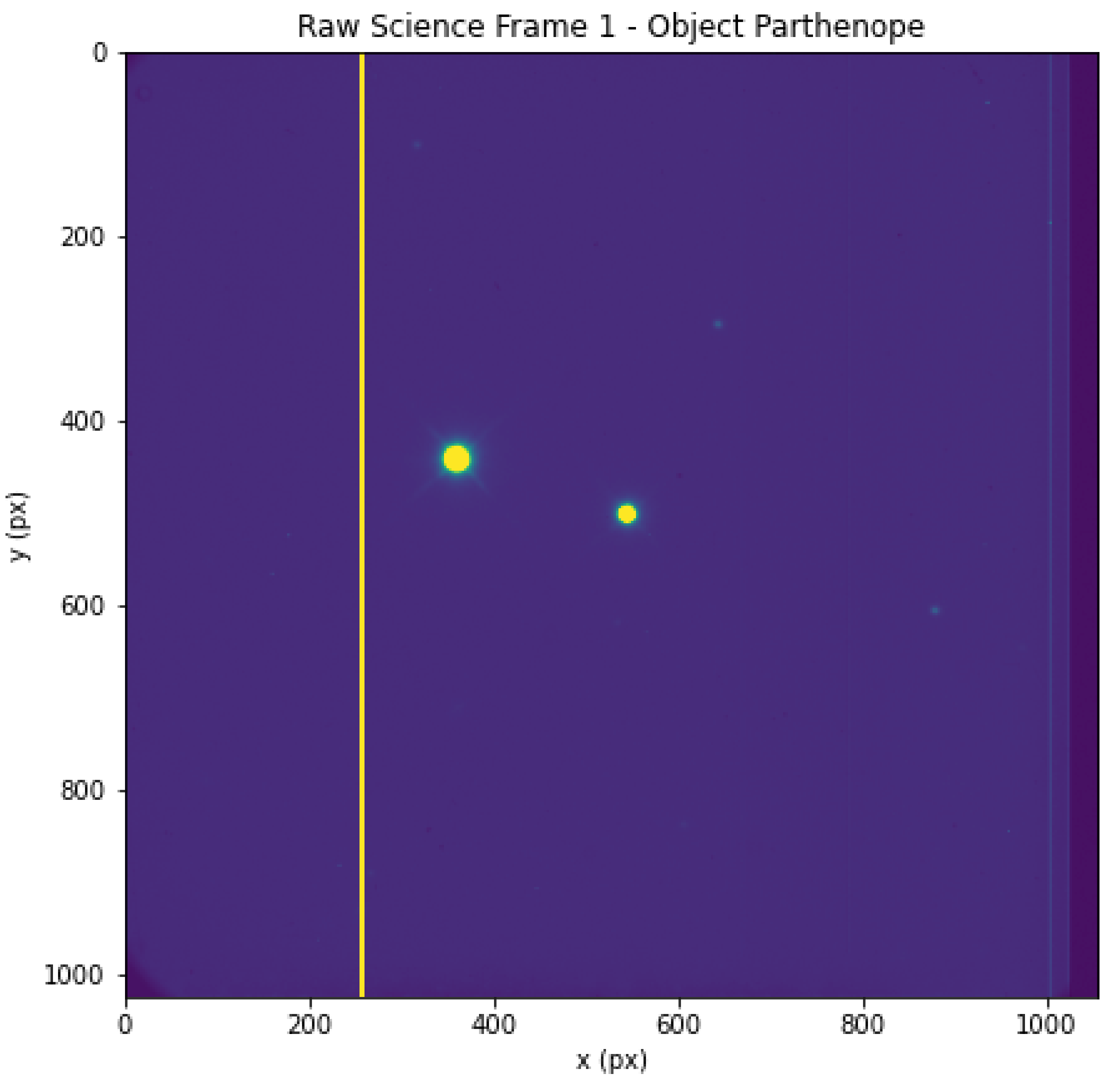
[8]:
```
def load_reduced_science_frame1(filename):

    #Loads science frame and reduces it via subtracting the bias and normalized␣
↪flat field correction

    science = fits.getdata(data_dir+data_files[2])[:,:1024]-fits.
↪getdata(data_dir+data_files[0])[:,:1024]+1.000000001
    flat = fits.getdata(data_dir+data_files[1])[:,:1024]-fits.
↪getdata(data_dir+data_files[0])[:,:1024]+1.000000001

    normflat = flat/(np.mean(flat))

    data = science/normflat
```

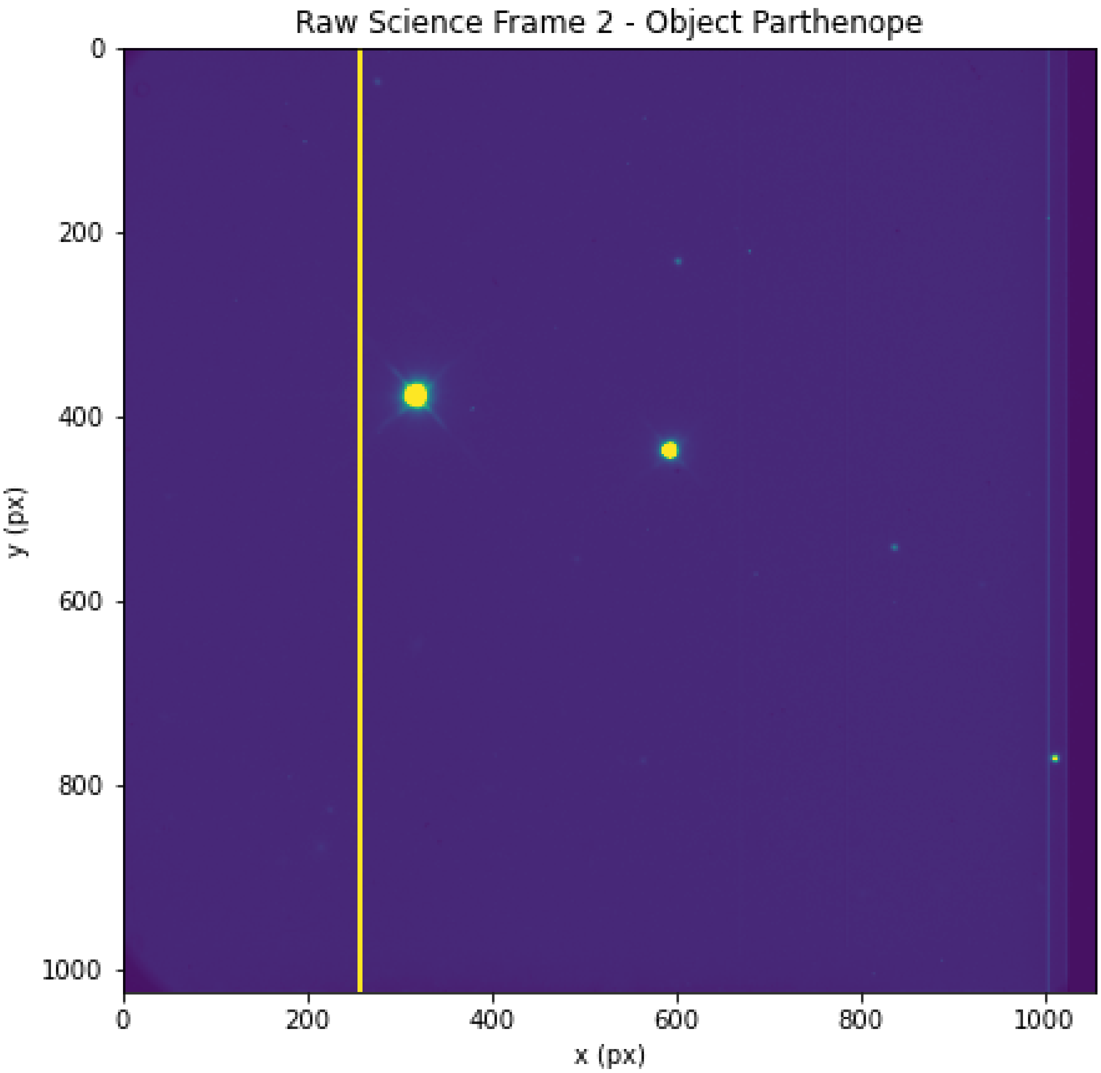
```
    return data

def load_reduced_science_frame2(filename):

    #Was having an issue with duplicate images, so two reduction functions was␣
 ↪my frustrated solution

    science = fits.getdata(data_dir+data_files[4])[:,:1024]-fits.
 ↪getdata(data_dir+data_files[0])[:,:1024]+1.000000001
    flat = fits.getdata(data_dir+data_files[1])[:,:1024]-fits.
 ↪getdata(data_dir+data_files[0])[:,:1024]+1.000000001

    normflat = flat/(np.mean(flat))

    data = science/normflat

    return data
```

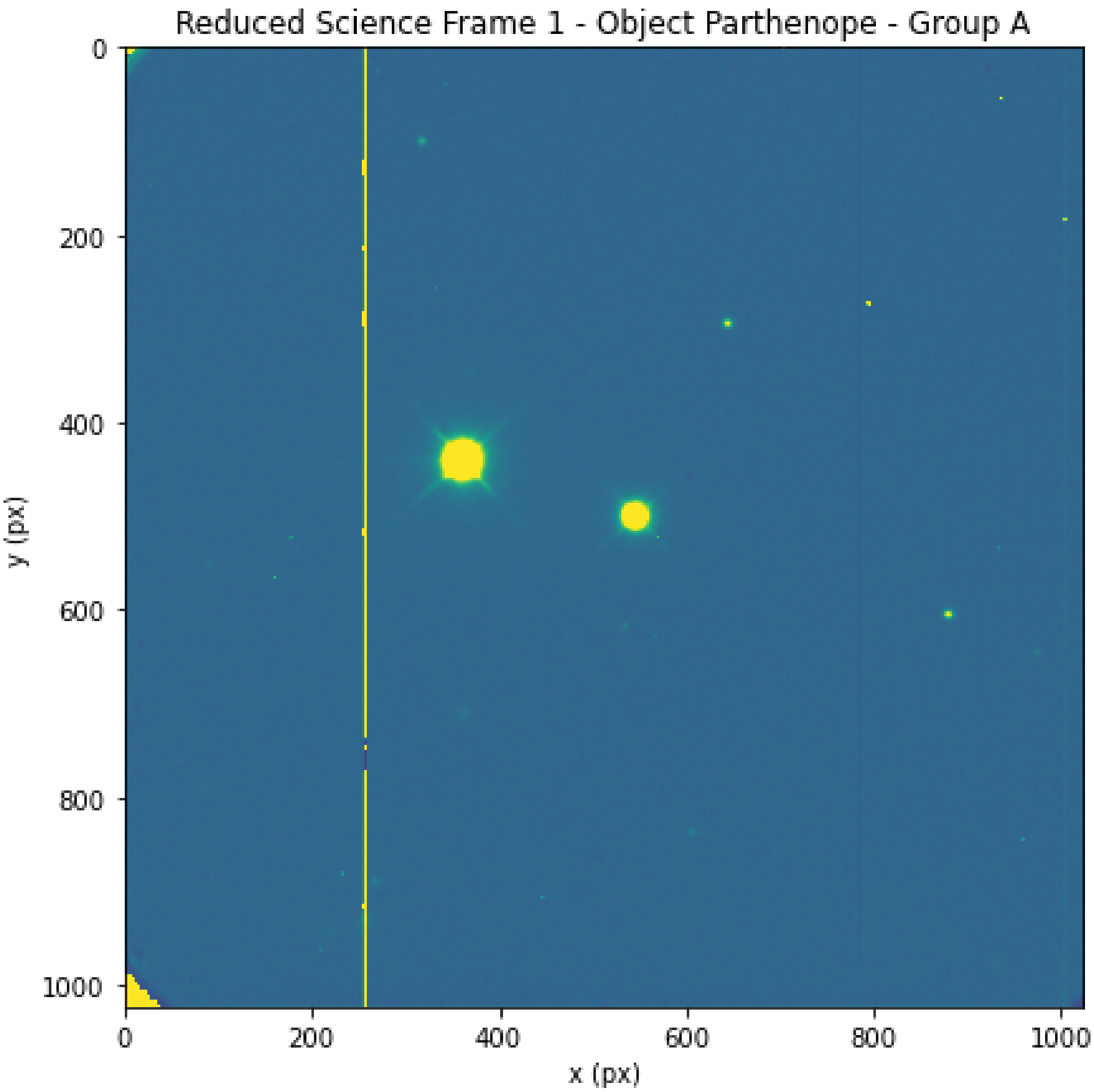
[9]:
```
#First and second science frames of Parthenope

sci1 = 'd40.fits'

sci2 = 'd90.fits'
```

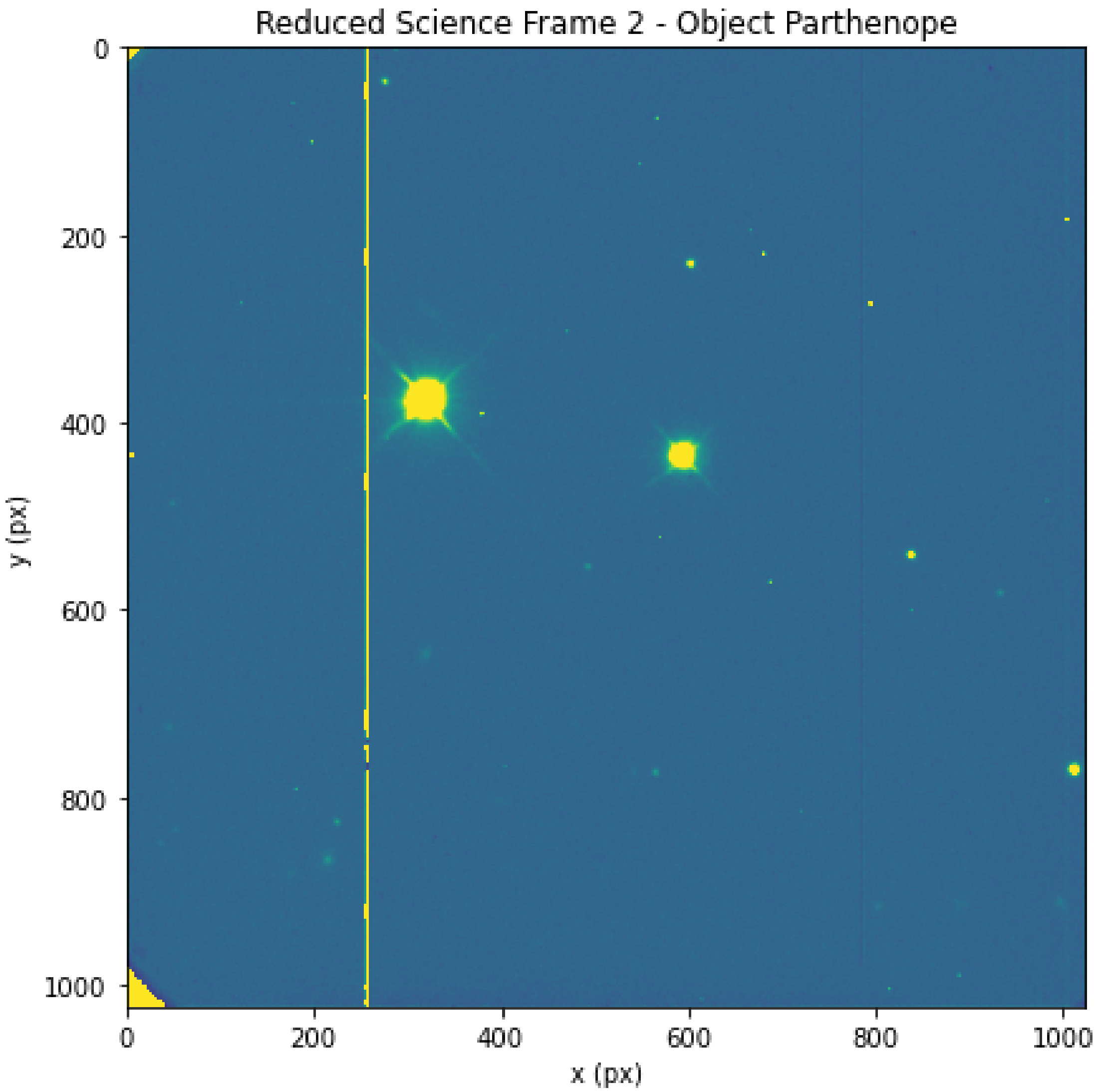
[32]:
```
#Plotting the science frames before and after they are reduced
#Before:
data3 = fits.getdata(data_dir+data_files[2])

fig2, ax2 = plt.subplots(figsize = [7, 7])
plot_fits_im(ax2, data3,
             title='Raw Science Frame 1 - Object {}'.format(objects[data_files␣
 ↪== sci2][0]),
             vmax = np.median(data3) * 3, cmap = "viridis")

data4 = fits.getdata(data_dir+data_files[4])

fig2, ax2 = plt.subplots(figsize = [7, 7])
plot_fits_im(ax2, data4,
             title='Raw Science Frame 2 - Object {}'.format(objects[data_files␣
 ↪== sci2][0]),
             vmax = np.median(data4) * 3, cmap = "viridis")


#After:
data1 = load_reduced_science_frame1(sci1)
```

```
fig2, ax2 = plt.subplots(figsize = [7, 7])
plot_fits_im(ax2, data1,
             title='Reduced Science Frame 1 - Object {}'.
 ↪format(objects[data_files == sci1][0]),
             vmax = np.median(data1) * 3, cmap = "viridis")
fig2.show()

data2 = load_reduced_science_frame2(sci2)

fig2, ax2 = plt.subplots(figsize = [7, 7])
plot_fits_im(ax2, data2,
             title='Reduced Science Frame 2 - Object {}'.
 ↪format(objects[data_files == sci2][0]),
             vmax = np.median(data2) * 3, cmap = "viridis")
fig2.show()
```




Raw Science Frame 2 - Object Parthenope
y (px)
x (px)
0
200
400
600
800
1000
0
200
400
600
800
1000


Reduced Science Frame 1 - Object Parthenope - Group A
y (px)
x (px)
0
200
400
600
800
1000
0
200
400
600
800
1000



0.4 Identifying Stars

[11]:
```
#Finds star locations
def find_star_locs(im_data, n_size = 10, bright_count_thresh=30):

    #set bright threshhold and neighbourhood size
    bright_thresh = np.median(im_data) * 1.5
    bright_pixels = np.array(np.where(im_data > bright_thresh)).T
    list_pos = []
    n_size = 10

    for bright_pixel in bright_pixels:
```

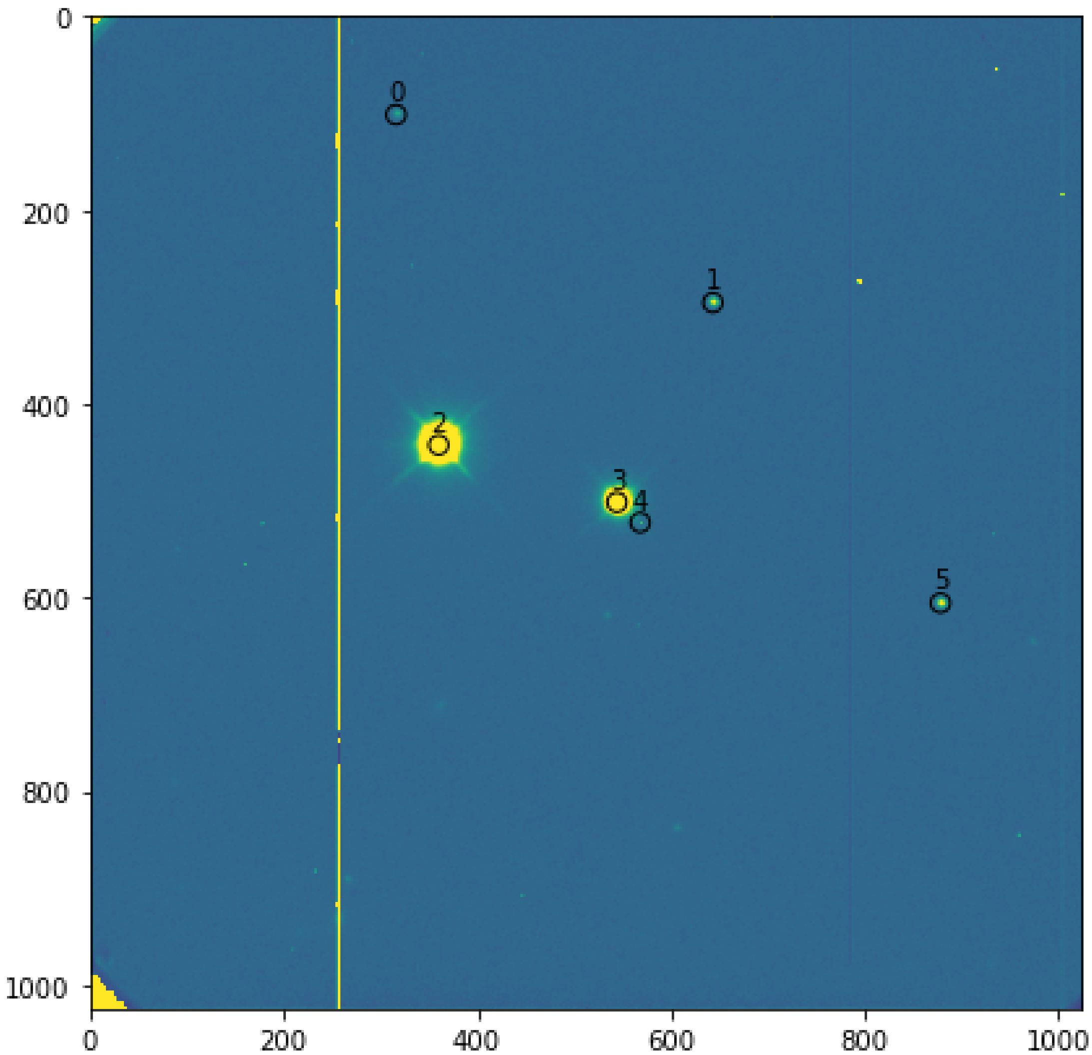
```
        neighbourhood = im_data[bright_pixel[0] - n_size:bright_pixel[0] +␣
↪n_size, bright_pixel[1] - n_size:bright_pixel[1] + n_size]
    #If too close to the edge, we assume it isn't a star
        if np.size(neighbourhood) == 0:
            continue

            #Must exceed a certain number of bright pixels to be a star
        bright_count = len(neighbourhood[neighbourhood > bright_thresh])
        if bright_count < bright_count_thresh:
            continue

            #Computes width and height of a star
        n_bright_pixels = np.where(neighbourhood > bright_thresh)
        width = max(n_bright_pixels[0]) - min(n_bright_pixels[0])
        height = max(n_bright_pixels[1]) - min(n_bright_pixels[1])

        #If the shape of the object is too weird, it's probably not a star
        star_shape = min(width, height) / float(max(height, width))
        if star_shape < (1.0 / 2.0):
            continue

        if (im_data[bright_pixel[0], bright_pixel[1]] >= np.max(neighbourhood)):
            list_pos += [[bright_pixel[0], bright_pixel[1]]]

    return list_pos
```

[12]:

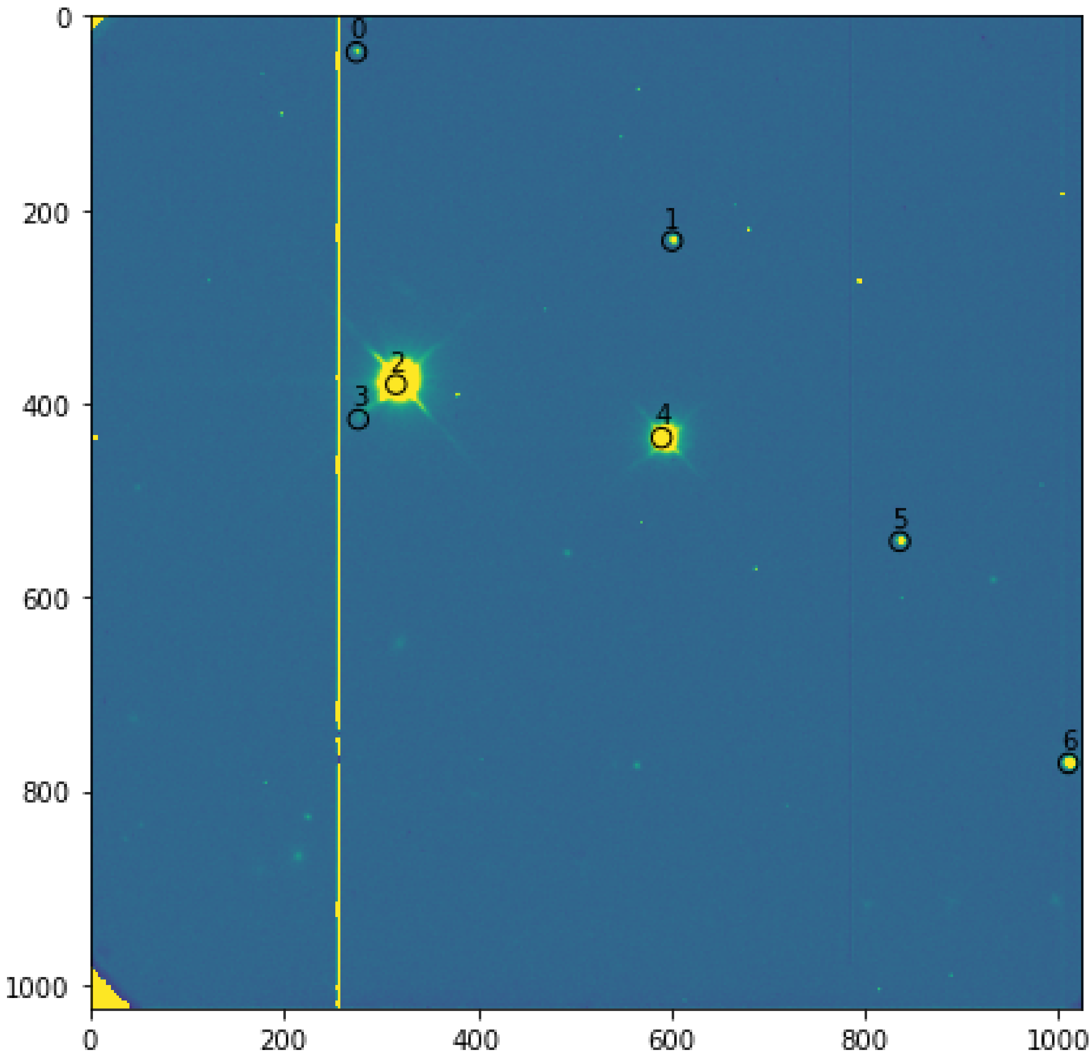
```
#Plots circles around star locations on the two reduced science frames
ident_star_locs1 = find_star_locs(data1, bright_count_thresh=8)

import matplotlib.patches as patches
plt.figure(figsize = [7, 7])
plt.imshow(data1, vmax = np.median(data1) * 3, cmap = "viridis")

for i, ident_star_locs in enumerate(ident_star_locs1):
    plt.axes().add_patch(patches.Circle([ident_star_locs[1],␣
↪ident_star_locs[0]], 10, fill=False, color = 'black'))
    plt.text(ident_star_locs[1] - 8, ident_star_locs[0] - 15, str(i), color =␣
↪'black')

plt.show()

ident_star_locs2 = find_star_locs(data2, bright_count_thresh=20)

import matplotlib.patches as patches
plt.figure(figsize = [7, 7])
plt.imshow(data2, vmax = np.median(data2) * 3, cmap = "viridis")
```

```
for i, ident_star_locs in enumerate(ident_star_locs2):
    plt.axes().add_patch(patches.Circle([ident_star_locs[1],␣
 ↪ident_star_locs[0]], 10, fill=False, color = 'black'))
    plt.text(ident_star_locs[1] - 8, ident_star_locs[0] - 15, str(i), color =␣
 ↪'black')

plt.show()
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:9:
MatplotlibDeprecationWarning: Adding an axes using the same arguments as a
previous axes currently reuses the earlier instance.  In a future version, a new
instance will always be created and returned.  Meanwhile, this warning can be
suppressed, and the future behavior ensured, by passing a unique label to each
axes instance.
  if __name__ == '__main__':
```



```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:21:
MatplotlibDeprecationWarning: Adding an axes using the same arguments as a
previous axes currently reuses the earlier instance.  In a future version, a new
instance will always be created and returned.  Meanwhile, this warning can be
suppressed, and the future behavior ensured, by passing a unique label to each
axes instance.
```



0.5 Calculating the Star Centroids

[13]:
```
#Function that calculates the centroids of the stars and returns their x and y␣
↪coordinates
def opcentroidf(arr,locs):
```

```
    i = 0
    median = np.median(arr)
    cx = []
    cy = []
    cxy = []
    cxe = []
    cye = []
    cents = []

    while i<len(locs):
        j = 0
        while arr[locs[i][0]+j][locs[i][1]] > median:
            j+=1
        if j > 30:
            j = 30
        x = locs[i][0]
        y = locs[i][1]

        s_intensity = np.sum(arr[x-j:x+j,y-j:y+j])

        k = 0
        xs = 0
        ys = 0

        while k<2*j:
            ys+=np.sum(np.multiply(np.arange(x-j,x+j),arr[x-j:x+j,y-j+k]))
            xs+=np.sum(np.multiply(np.arange(y-j,y+j),arr[x-j+k,y-j:y+j]))

            k+=1

        cy = (ys/s_intensity)
        cx = (xs/s_intensity)

        cents.append([cy,cx])

        i+=1

    return cents
```

[14]:
```
#Returns both the centroid coordinates and the star locations for comparison
cents1=opcentroidf(data1,ident_star_locs1)
print("Centroids:\n",cents1,"\n")
print("Star Locations:\n",ident_star_locs1,"\n")
print("Difference:\n",np.abs(np.subtract(cents1,ident_star_locs1)),"\n")
cents2=opcentroidf(data2,ident_star_locs2)
print("Centroids:\n",cents2,"\n")
```

```
print("Star Locations:\n",ident_star_locs2,"\n")
print("Difference:\n",np.abs(np.subtract(cents2,ident_star_locs2)),"\n")
```

```
Centroids:
 [[100.45701623819133, 315.6210525556931], [294.496987372875,
642.5271416913764], [440.6843999831248, 358.88320431916577],
[500.13565163060946, 543.4016225491586], [520.738361955559, 566.6831255824168],
[604.529780706604, 877.516019596701]]

Star Locations:
 [[101, 316], [295, 643], [442, 360], [501, 544], [522, 568], [605, 878]]

Difference:
 [[0.54298376 0.37894744]
 [0.50301263 0.47285831]
 [1.31560002 1.11679568]
 [0.86434837 0.59837745]
 [1.26163804 1.31687442]
 [0.47021929 0.4839804 ]]

Centroids:
 [[36.481087235435076, 274.6336046013176], [231.32051968589883,
600.6721994831756], [377.058633944083, 316.93854514579044], [415.35247097184777,
276.5331089353196], [435.8751467244046, 591.4180544889977], [541.3827074642261,
835.6550434605468], [770.3345641542171, 1009.5787726161514]]

Star Locations:
 [[37, 275], [232, 601], [380, 316], [416, 277], [435, 590], [542, 836], [771,
1010]]

Difference:
 [[0.51891276 0.3663954 ]
 [0.67948031 0.32780052]
 [2.94136606 0.93854515]
 [0.64752903 0.46689106]
 [0.87514672 1.41805449]
 [0.61729254 0.34495654]
 [0.66543585 0.42122738]]
```

[15]:

```
#Finds the error on each centroid for both of the data sets, and then
 ↪calculates the average percent error for both.
def ecent(arr,cents):
    i=0
    median = np.median(arr)
    e = []
    while i <len(cents):
```

```
        j = 10
        x = int(cents[i][0])
        y = int(cents[i][1])

        ei = np.std(arr[x-j:x+j,y-j:y+j])
        es = np.sum(arr[x-j:x+j,y-j:y+j])

        e.append(ei/es)
        i+=1
    return e

ecents1 = ecent(data1,cents1)
ecents2 = ecent(data2,cents2)
print("Percent error on each centroid for data1: ",ecents1,"\nAverage Percent␣
 ↪Error:",np.mean(ecents1)*100)
print("\nPercent error on each centroid for data2:",ecents2,"\nAverage Percent␣
 ↪Error:",np.mean(ecents2)*100)
```

```
Percent error on each centroid for data1:  [0.0004873769648777923,
0.0009737514038465543, 0.0021300450205492135, 0.003583052956702228,
0.0007822933055267341, 0.0010522355164532178]
Average Percent Error: 0.15014591946592898

Percent error on each centroid for data2: [0.00089742391483719,
0.0017084749481659238, 0.00265602127645748, 0.0004439431380934641,
0.004286014209127055, 0.001822532812820504, 0.0035414405490488757]
Average Percent Error: 0.21936929783643563
```

0.6 Retrieving Star Positions from USNO-B1 Catalog for field

[16]:
```
from astroquery.simbad import Simbad
from astroquery.vizier import Vizier
from astropy.coordinates import SkyCoord
import astropy.units as u



h = fits.open(data_dir+data_files[2])
science_ra_center = h[0].header["RA"]
science_dec_center= h[0].header["DEC"]
h.close()

print("Parthenope Group A (data_file 1)")
print("RA  of science frame {}: {} radians".format(data_files[2],␣
 ↪science_ra_center ))
print("Dec of science frame {}:{} radians". format(data_files[2],␣
 ↪science_dec_center))
```

```
h = fits.open(data_dir+data_files[4])
science_ra_center = h[0].header["RA"]
science_dec_center= h[0].header["DEC"]
h.close()

print("Parthenope (data file 2)")
print("RA  of science frame {}: {} radians".format(data_files[4],␣
 ↪science_ra_center ))
print("Dec of science frame {}:{} radians". format(data_files[4],␣
 ↪science_dec_center))
```

```
Parthenope Group A (data_file 1)
RA  of science frame d40.fits: 01:40:43.09 radians
Dec of science frame d40.fits:02:42:57.6 radians
Parthenope (data file 2)
RA  of science frame d90.fits: 01:40:42.99 radians
Dec of science frame d90.fits:02:41:58.0 radians
```

[17]:
```
ra_center = '01:40:35.00'
dec_center = '02:49:00.00'

center_coord = SkyCoord(ra = ra_center, dec = dec_center, unit = (u.hour, u.
 ↪deg), frame='icrs')
```

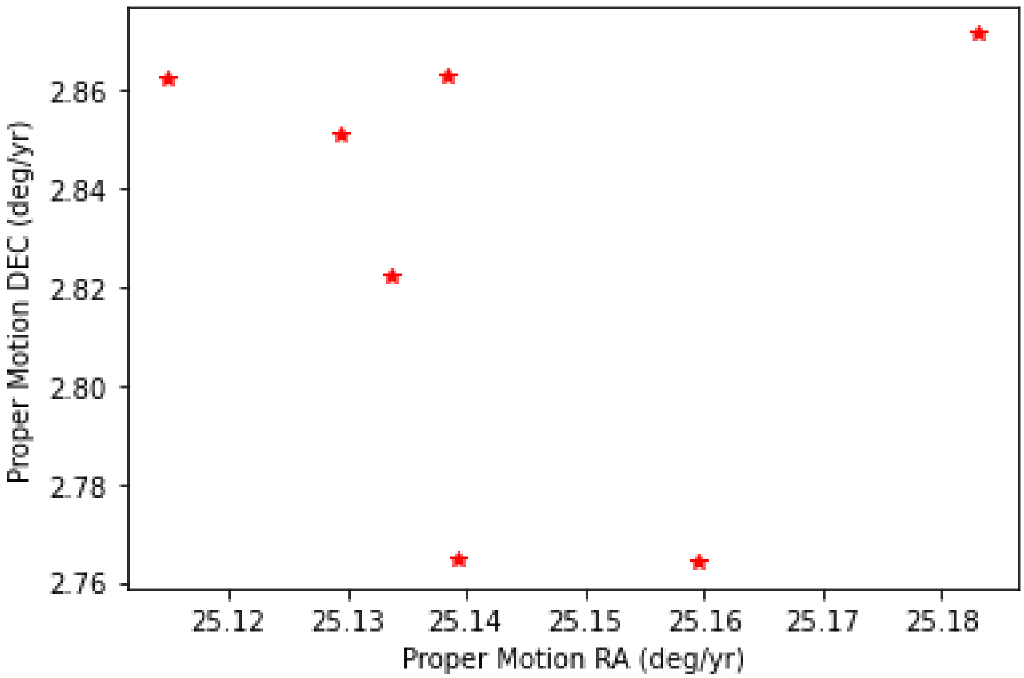
[18]:
```
#Setting a limit for star brightness
magnitude_limit = 16.2

# Create a VizieR object.
vizier = Vizier(column_filters = {"R2mag" : "<{}".format(magnitude_limit)})

# By default, VizieR will limit the result to 50 stars. We change the limit to␣
 ↪"-1" meaning "show everything you have"
# This is a dangerous setting, since some queries may return enormous amounts␣
 ↪of data. In our case, however, only a small
# part of the sky is being queried, heavily limited by magnitude. We should be␣
 ↪safe.
vizier.ROW_LIMIT = -1

# Run the query. We are requesting the part of the sky defined by the␣
 ↪previously created coordinates object (center).
# width, height specify the size of the region we need (in arc minutes).
# Adjust this value to match your science frame
# "catalog = "USNO-B1" tells VizieR which catalogue we want. This could be a␣
 ↪list of multiple catalogues.
```

```
fov_width  = '7.0m'
fov_height = '7.0m'

result_table = vizier.query_region(center_coord, width = fov_width, height =␣
 ↪fov_height, catalog = "USNO-B1")

print ("Total number of objects retrieved:", len(result_table[0]))
print ("Column names:", result_table[0].keys())
```

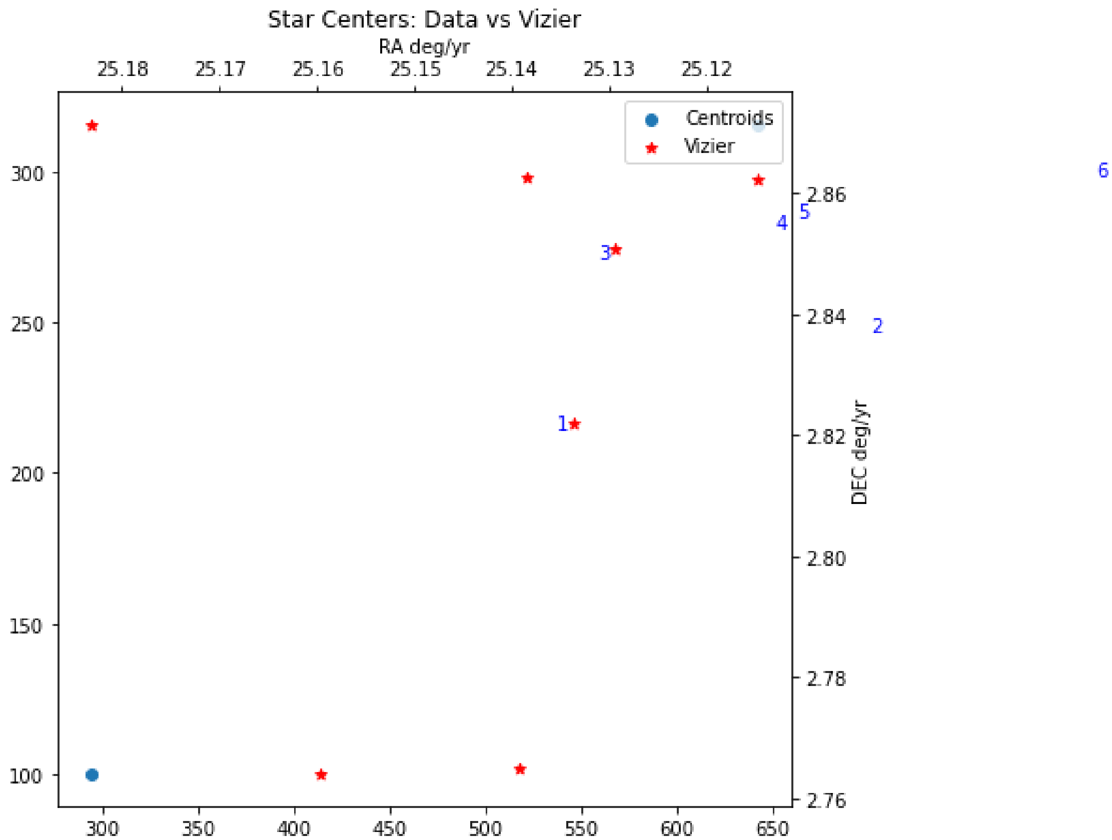
```
Total number of objects retrieved: 7
Column names: ['USNO-B1.0', 'RAJ2000', 'DEJ2000', 'e_RAJ2000', 'e_DEJ2000',
'Epoch', 'pmRA', 'pmDE', 'Ndet', 'B1mag', 'R1mag', 'B2mag', 'R2mag', 'Imag']
```

[19]:
```
ra_cat     = np.array(result_table[0]["RAJ2000"])
dec_cat    = np.array(result_table[0]["DEJ2000"])
pm_ra      = np.array(result_table[0]["pmRA"])
pm_dec     = np.array(result_table[0]["pmDE"])
mag        = np.array(result_table[0]["R2mag"])

epoch = 2020.0   # Image taken in 2020

# Convert the units of pmDec from mas/yr to deg/yr
pm_dec = pm_dec / 1000.0 / 3600.0

# Same for RA, but with the cos(dec) correction. Note, np.cos() expects radians!
pm_ra  = pm_ra / 1000.0 / 3600.0 / np.cos(dec_cat * np.pi / 180.0)

# Move the stars
ra_cat  += (epoch - 2000.0) * pm_ra
dec_cat += (epoch - 2000.0) * pm_dec

plt.xlabel("Proper Motion RA (deg/yr)")
plt.ylabel("Proper Motion DEC (deg/yr)")
plt.plot(ra_cat, dec_cat, 'r*')
```

[19]:
```
[<matplotlib.lines.Line2D at 0x7fd3322ccd10>]
```

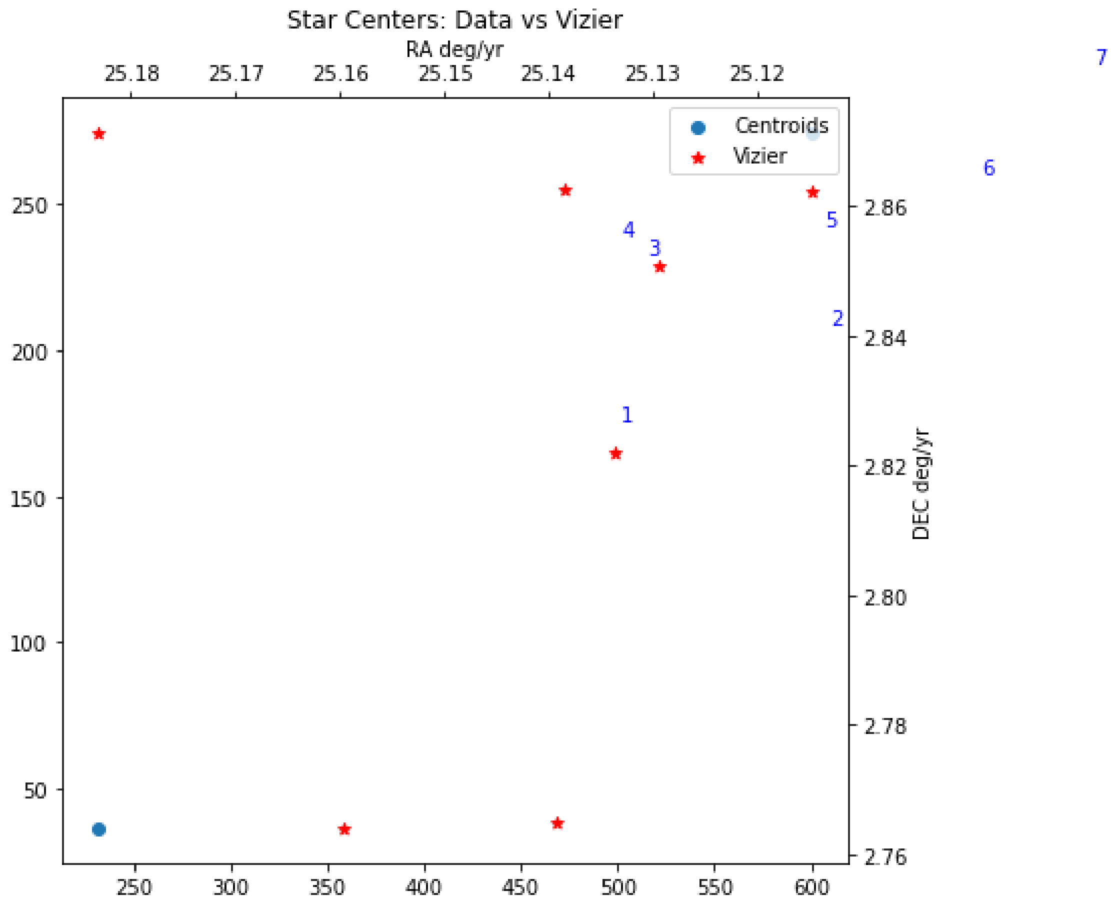


[20]:
```python
#Science frame 1
fig4 = plt.figure(figsize = [7, 7])
axs = fig4.add_subplot(111, label='science', frame_on=True)
axq = fig4.add_subplot(111, label='Query'  , frame_on=False)

science_points = axs.scatter(np.squeeze(cents1[1]), np.squeeze(cents1[0]))
queried_points = axq.scatter(ra_cat, dec_cat, color='red', marker='*')

# Not flipping axis, worked much better without it

for i, (loc_y, loc_x) in enumerate(cents1):
    axs.text((loc_x)/2 + 380, (loc_y)/6 + 198, str(i+1), color = 'blue')

axq.xaxis.tick_top()
axq.xaxis.set_label_position('top')
axq.yaxis.tick_right()
axq.yaxis.set_label_position('right')
axq.set_xlim(axq.get_xlim()[::-1])

plt.title("Star Centers: Data vs Vizier")
plt.legend([science_points,queried_points],["Centroids","Vizier"])
plt.xlabel("RA deg/yr")
plt.ylabel("DEC deg/yr")

fig4.show()
```




```
[21]: #Science Frame 2
fig4 = plt.figure(figsize = [7, 7])
axs = fig4.add_subplot(111, label='science', frame_on=True)
axq = fig4.add_subplot(111, label='Query'   , frame_on=False)

science_points = axs.scatter(np.squeeze(cents2[1]), np.squeeze(cents2[0]))
queried_points = axq.scatter(ra_cat, dec_cat, color='red', marker='*')

# Not flipping axis, worked much better without it

for i, (loc_y, loc_x) in enumerate(cents2):
    axs.text((loc_x)/3 + 410, (loc_y)/6 +170, str(i+1), color = 'blue')

axq.xaxis.tick_top()
axq.xaxis.set_label_position('top')
axq.yaxis.tick_right()
axq.yaxis.set_label_position('right')
axq.set_xlim(axq.get_xlim()[::-1])

plt.title("Star Centers: Data vs Vizier")
```

```
plt.legend([science_points,queried_points],["Centroids","Vizier"])
plt.xlabel("RA deg/yr")
plt.ylabel("DEC deg/yr")

fig4.show()
```



0.7 Solving for Plate Constants

[22]:
```
def sin(angle):
    """ Converts to degrees then does the sin """
    return np.sin(angle * np.pi / 180.0)

def cos(angle):
    """ Converts to degrees then does cos """
    return sin(angle + 90)

bottom = cos(center_coord.dec.deg) * cos(dec_cat) * cos(ra_cat - center_coord.
 ↪ra.deg) + sin(center_coord.dec.deg) * sin(dec_cat)
```

```
X = - cos(dec_cat) * sin(ra_cat - center_coord.ra.deg) / bottom
Y = - (sin(center_coord.dec.deg) * cos(dec_cat) * cos(ra_cat - center_coord.ra.
 ↪deg) - cos(center_coord.dec.deg) * sin(dec_cat)) / bottom

f = 16480
p = 0.015

fp = f/p
#The following would be done if we had an ideal camera. Unfortunately, we have␣
 ↪shear and distortion.
#x = f * (X/p) +np.shape(data1)[1] / 2.0
#y = f * (Y/p) +np.shape(data1)[0] / 2.0
```

[23]:
```
#Matrix math for solving for plate constants
B = np.array([
    [fp *X[0],fp * Y[0],1],
    [fp *X[1],fp * Y[1],1],
    [fp *X[2],fp * Y[2],1],
    [fp *X[3],fp * Y[3],1],
    [fp *X[4],fp * Y[4],1],
    [fp *X[5],fp * Y[5],1],
])

a1 = np.array([
    cents1[0][0],
    cents1[1][0],
    cents1[2][0],
    cents1[3][0],
    cents1[4][0],
    cents1[5][0],
])

a2 = np.array([
    cents1[0][1],
    cents1[1][1],
    cents1[2][1],
    cents1[3][1],
    cents1[4][1],
    cents1[5][1],
])

c1 = np.matmul(np.matmul(inv(np.matmul(np.transpose(B),B)),np.transpose(B)),a1)
c2 = np.matmul(np.matmul(inv(np.matmul(np.transpose(B),B)),np.transpose(B)),a2)
print(c1)
print(c2)
```

```
[-3.07405165e-01 -1.45738058e-01  5.08229830e+02]
```

```
[-1.86117545e-01 -1.80003190e-01  6.49459623e+02]
```

[24]:
```
#Final steps
a = [fp * c1[0], fp * c1[1], c1[2]]
b = [fp * c2[0], fp * c2[1], c2[2]]
T = np.array([a,b,[0,0,1]])
x_coords = [cents1[3][0], cents2[3][0]]
y_coords = [cents1[4][1], cents2[4][1]]
X1 = np.matmul(inv(T),np.transpose([x_coords[0],y_coords[0],1.0]))
X2 = np.matmul(inv(T),np.transpose([x_coords[1],y_coords[1],1.0]))
print(X1)
print(X2)
print(center_coord)
```

```
[-3.42230940e-04  7.72418909e-04  1.00000000e+00]
[2.66492385e-04 1.79451762e-05 1.00000000e+00]
<SkyCoord (ICRS): (ra, dec) in deg
    (25.14583333, 2.81666667)>
```

0.8 Calculating Proper Motion of Parthenope

[25]:
```
#Calculating proper motion via distance formula
ra0 = center_coord.dec.deg
dec0 = center_coord.ra.deg

ra_i = np.arctan(-X1[0]/(cos(dec0)-X1[1]*sin(dec0))) + (ra0*np.pi/180)
dec_i = np.arcsin((sin(dec0)+X1[1]*cos(dec0))/np.sqrt(1+X1[0]**2 + X1[1]**2))

ra_f = np.arctan(-X2[0]/(cos(dec0)-X2[1]*sin(dec0))) + (ra0*np.pi/180)
dec_f = np.arcsin((sin(dec0)+X2[1]*cos(dec0))/np.sqrt(1+X2[0]**2 + X2[1]**2))

print(ra_i,dec_i)
print(ra_f,dec_f)
dt = 2.15 #hrs

vra = (ra_f - ra_i)/dt * 180/np.pi
vdec = (dec_f - dec_i)/dt * 180/np.pi

error_p = (np.mean(ecents1) + np.mean(ecents2))/2
print(" ")
print("RA Velocity (arcsec/hr):", vra * 60)
print("RA Velocity error: ", abs(vra * 60 * error_p))
print(" ")
print("DEC Velocity (arcsec/hr): ", vdec * 60)
print("DEC Velocity error: ", abs(vdec * 60 * error_p))
```

```
0.04953830437152004 0.43964997603573414
0.04886571261874219 0.43889551333151267
```

```
RA Velocity (arcsec/hr): -1.0754419191482096
RA Velocity error:  0.0019869607722506136

DEC Velocity (arcsec/hr):  -1.206349639588387
DEC Velocity error:  0.00222882274607566
```

This has been by far the most frustrating and difficult lab to complete. I spent hours trying to piece it all together, along with my lab partner, Lucas. Despite this, I've had a great time and learned an immense amount from these labs and this class. Thank you