Observational Astrophysics Lab
Lab Report #3

ASTROMETRY FROM CCD IMAGES

March 8, 2022

Ethan Wadhwa, Group CD
ewadhwa@ucsd.edu
PID: A15778800

Professor: Dr. Quinn Konopacky
TA: Clarissa Do Ó

ABSTRACT

In this lab, we took CCD images of asteroids using the Direct Imaging Camera on the Nickel 1-m Telescope at the Lick Observatory. We first developed algorithms to reduce our science frame by subtracting the bias and overscan area, multiplying by the gain obtained from flatfield images. We then developed an algorithm to identify and centroid star locations to accurately chart star locations on our CCD images. We take the star locations from our image and query USNO-B1 to find the actual right ascension and declination of the stars in our frame. We then wrote an algorithm that takes the actual star right ascension and declination and star locations in our frame to convert our frames from pixels to standard coordinates using. Once the objects in our frames have been converted into standard coordinates, we can relate the locations of the observed asteroid across frames to find the velocity of the asteroid.

1 Introduction

In astronomy, the celestial sphere is a tool used to chart the locations of celestial objects in the sky. The celestial sphere is a fixed projection of the sky onto a sphere with its equator along the same axis as the Earths. By charting celestial objects on this sphere, astronomers label objects' coordinates on the fixed sphere in terms of their right ascension (degrees to the left or right of the celestial meridian) and declination (degrees above or below the celestial equator).
Star fields are the backgrounds of most astronomical images taken; they are the fields of visible stars in the area behind a source being observed. Star fields are used for a much more important task than being background noise behind a desired source; they allow astronomers to find the exact location of a source on the celestial sphere. The star field in an image of a source can be used to query databases containing the locations of known objects on the celestial sphere to convert the image of a source from pixels to celestial coordinates. By doing this, astronomers can accurately chart the motions of astronomical bodies by their movement along the celestial sphere.

2 Observations & Data

2.1 Collecting Data

In this lab, we collected our CCD images using the Direct Imaging Camera on the Nickel 1-m Telescope at the Lick Observatory. The telescope was accessed remotely by Dr. Quinn Konopacky and her physics 164 students on February 28th, 2022 from the UCSD Observational Astrophysics Lab Room. Several bias frames with the telescope shudder closed were taken, as well as dome flatfield images taken in each of the five possible filters (UBVRI). The main data used for this lab were CCD images taken by the physics 164 students of their chosen asteroids. For our group, C/D, we chose to observe 44 Nysa, so three sets of five CCD images of 44 Nysa were taken at different times throughout the observing period, two sets in the R filter and one set taken in the V filter. The full list of data files used can be found in appendix 6.1.

2.2 Catalog Databases

In this lab we rely on the USNO-B1 database to chart the predicted locations of stars in our frames' star fields. The USNO-B1 is a catalog of ~1 billion celestial objects derived from ~3.6 billion observations taken by the US Naval Observatory (USNO), all with magnitudes less than V=21. This catalog also contains the approximate motion of the charted celestial objects to an accuracy of approximately 0.2 arc seconds to the star's locations in the year 2000. Movement of each celestial body per year or 'epoch' has been calculated in this catalog as well, allowing us to use the predicted locations of stars on our observing data (02/28/2022) to relate our CCD images to actual celestial coordinates.

2.3 Displaying Data

Let us take a closer look at the first and last 44 Nysa frames taken in the R band. To be able to properly view our CCD images, we must first perform a bias subtraction, overscan removal, and flatfield correction to display our images in their clearest form. Once this has been done, we can observe several frames to view the movement of our chosen asteroid over time. As seen in figure 1 below, the chosen object (44 Nysa) clearly moves a significant distance within the time between these two frames being taken. Additionally, in this image we can see the same star field in the background of both frames, telling us we can use these stars to find the exact right ascension and declination of 44 Nysa at the times of these frames.

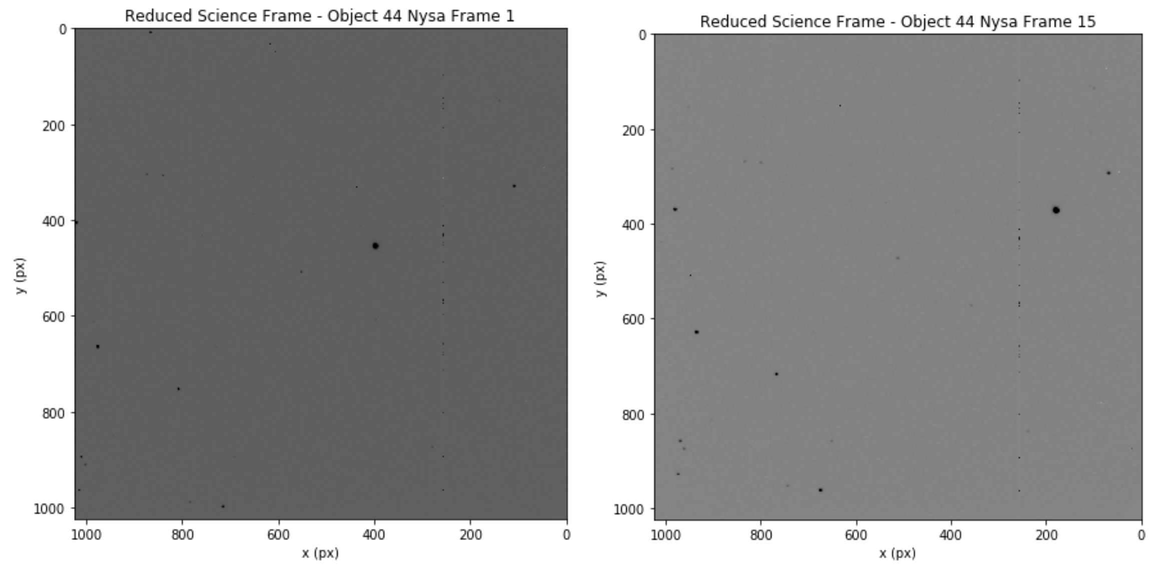

Reduced Science Frame - Object 44 Nysa Frame 1
Reduced Science Frame - Object 44 Nysa Frame 15
y (px)
x (px)
0
200
400
600
800
1000

Figure 1: The first (left) and last (right) reduced 44 Nysa frames shown with gray color scale plotting to be able to better visualize objects present in the frame. The darker the points, the more intensity observed on those pixels. A bad column of detectors is present at roughly x=252, causing the inconsistency along that column. The largest dark point visible in both images is 44 Nysa.

3 Data Reduction & Methods

3.1 Reduction of Astronomical Science Frames

The first step in reducing our data to be usable for our calculations is to remove the bias noise and overscan region, and to correct for pixel sensitivity using dome flatfields. For this dataset, we obtain a series of bias frames which can be averaged to use as a universal bias frame, and the overscan region is given in the header of each file. To make this process simpler, we first remove the bias and overscan from all frames before using the normalized flatfield frame to correct for pixel sensitivity in our science frames. To remove the overscan and bias from our frames, we subtract the averaged bias frame from our science frames:

$$Frame_{bias\ sub} = Frame - Bias_{avg}$$

Then we remove the columns of the dataset in the overscan region of the frame.

$$Frame_{bias\ sub\ overscan\ rem} = Frame_{bias\ sub}[:][0: len(xpixels) - overscan]$$

Where the first bracket represents the rows of the pixel values and the second bracket represents the columns of the pixel values. To correct for the gain in the pixel sensors using the normalized flatfield, we obtain the normalized flatfield by bias and overscan, correcting the flatfield frame with the same filter as our science frame and dividing the reduced flatfield frame by the median value of the reduced flatfield frame.

$$Gain = \frac{Flat_{avg_{bias\ sub\ overscan\ rem}}}{Median(Flat_{avg_{bias\ sub\ overscan\ rem}})}$$

This gives us our pixel gain, which we multiply our bias subtracted and overscan removed science frames by in order to obtain our fully reduced astronomical science frames. The reduced frames can be seen in Figure 1. The python functions used to reduce these CCD images are located in appendix 6.3.1, 6.3.2.

3.2 Identifying & Centroiding Star Locations

To convert from x pixels and y pixels to standard coordinates we must first define which points in our frame are stars. To do this, we locate potential stars by adding all pixels with analog-digital units (ADU) counts above 1.5 times the median ADU count of the frame to a list. To account for any pixels that detected low magnitude stars or that picked up light from any other source, we reduce our list of every pixel location that has a high intensity to only the pixels at the center of pixel clusters with high detected intensities. To do this, we search through a square of a defined size around each high intensity pixel and if there are less than a specified amount of other high intensity pixels in this area, we remove the pixel from our list. For pixels that do have more than the limit of nearby bright pixels, we identify the approximate location of the star to be at the highest ADU count pixel in the area. The code for this can be found at appendix 6.3.3.

By identifying general locations of stars, we are able to reduce our centroiding algorithm to just run in the areas of the general locations of the stars to find their exact centroids. We define the areas to perform a centroiding function to be a 20 pixel by 20 pixel square centered at the star locations identified on the frame. We then follow the centroiding algorithm for both x and y:

$$< x > = \frac{\sum_{i=1}^{N} x_i I_i}{\sum_{i=1}^{N} I_i}, \quad < y > = \frac{\sum_{i=1}^{N} y_i I_i}{\sum_{i=1}^{N} I_i} \tag{1}$$

Where x is the x pixel value at index i, y is the y pixel value at index i, and I is the ADU count (intensity) at index i. The centroid values of each star give us the calculated center of each star in x and y pixels with error calculated by:

$$\sigma_x = \frac{\sum_{i=1}^{N} (x_i - <x>)^2}{(\sum_{i=1}^{N} I_i)^2}, \quad \sigma_y = \frac{\sum_{i=1}^{N} (y_i - <y>)^2}{(\sum_{i=1}^{N} I_i)^2} \tag{2}$$

The centroiding function can be found at appendix 6.3.4, and is visualized in figure 2.

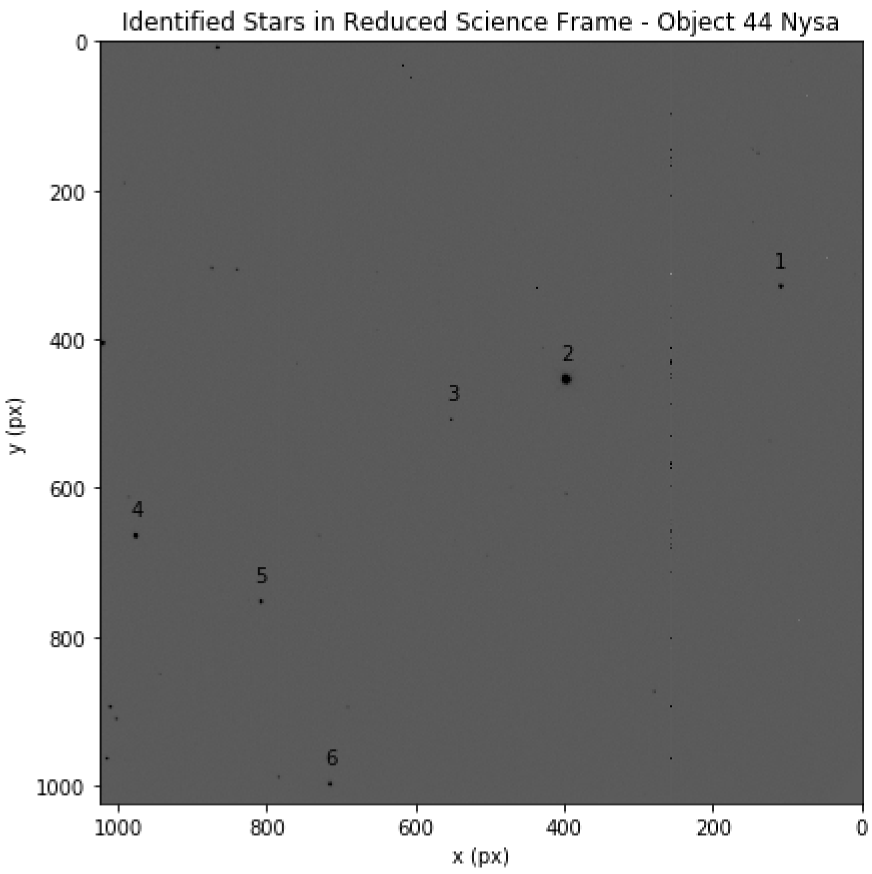


Figure 2: Identified Stars with counts over each star in the first 44 Nysa frame taken at 05:08:57.6, +20:58:51.4 at hour angle 21:00 on Feb 28, 2022. The brightest object marked with the number ‘2’ is 44 Nysa.

3.3 Querying USNO-B1 Database to find Star Field Coordinates

Once we have our star locations in pixel coordinates, the next piece of information is the actual right ascension and declination of the stars. To find the actual celestial coordinates of our stars, we must first find our frames' actual location on the celestial sphere. We find these coordinates by querying the USNO-B1 database with the right ascension and declination of the center of our frame obtained from the header of the data files. In order to access the USNO-B1 database, we import astroquery.vizier into our code. Vizier allows us to find the exact right ascension and declination of stars in the field of our image. We query Vizier with our frame coordinates, magnitude limit, number of stars, field of view dimensions, and year of observation. Vizier returns us a list of the celestial coordinates of any stars that meet our query condition. The code for querying USNO-B1 can be found at appendix 6.3.5.

3.4 Converting Frame Coordinates from Pixels to Standard Coordinates

Once we have obtained the pixel centroids of the stars in our frame and the queried actual right ascension and declination of each star, we can perform a fit using a least squares approximation of our data to convert our frame coordinates from pixels to celestial coordinates. To convert from x and y pixel values to RA and DEC, we rely on the matrix equation:

$$x = TX \tag{3}$$

where x is our centroid objects' pixel coordinate vector with the form (x , y , 1), X is our centroid objects' celestial coordinate vector with the form (RA, DEC, 1), and T is our transformation matrix:

$$\mathbf{T} = \begin{pmatrix} (f/p)a_{11} & (f/p)a_{12} & x_0 \\ (f/p)a_{21} & (f/p)a_{22} & y_0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4}$$

The (f/p) ratio in this matrix is the lens in the telescope's focal distance (f=16480 mm) and the pixel size (p=0.0015mm). The six unknowns in this equation are our plate constants, with $a_{i,j}$ referring to the scale, shear, and orientation and x_0, y_0 being our offset in pixels. We can find the values of the plate constants using least squares fitting. To find the plate constants, we must first write out our x=TX equation for the X matrix of the actual Ra and Dec of our stars and x being either the x pixel vector or y pixel vector. This gives us the matrix:

$$\mathbf{a} = \mathbf{Bc} \tag{5}$$

$$\mathbf{a} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} (f/p)X_1 & (f/p)Y_1 & 1 \\ (f/p)X_2 & (f/p)Y_2 & 1 \\ \vdots & \vdots & \vdots \\ (f/p)X_N & (f/p)Y_N & 1 \end{pmatrix}, \text{ and } \mathbf{c} = \begin{pmatrix} a_{11} & a_{12} & x_0 \end{pmatrix} \tag{6}$$

And the version of this matrix for y has the an a vector of all y pixels and c=(a_{21} a_{22} y_0)
Using these equations, we can calculate the plate constants by finding the least distance between the points through the equation:

$$\chi^2 = (\mathbf{a} - \mathbf{Bc})^T (\mathbf{a} - \mathbf{Bc}) \tag{7}$$

By minimizing χ^2, we can multiply both sides of a=Bc by the transpose of B and the inverse of B times transposed B to find our c vector values, giving us our plate constants.

$$\mathbf{c} = (\mathbf{B}^T \mathbf{B})^{-1} \mathbf{B}^T \mathbf{a} \tag{8}$$

By performing this fit using both a vectors (both x and y pixel coordinate vectors) to find both c vector values, we obtain our two sets of plate constants that we can plug back into matrix T. The equation to convert from RA/DEC to pixel coordinates can be reversed to convert from pixel values to ra/dec by multiplying both sides by the inverse of T to give us the equation:

$$X = T^{-1} x \tag{9}$$

We plug each individual pixel coordinate into this equation with the coordinate vector of the form x = [x pixel, y pixel, 1] to obtain our right ascension (α) and declination (δ) coordinates for our centroid object locations with the form X = [$\alpha,\ \delta,\ 1$]. Once we have obtained the celestial coordinates of our objects, it is a relatively straightforward conversion from celestial coordinates to standard coordinates:

$$X = -\frac{sin(\alpha-\alpha_0)cos(\delta)}{cos(\alpha-\alpha_0)cos(\delta)cos(\delta_0)+sin(\delta)sin(\delta_0)} \tag{10}$$

$$Y = -\frac{cos(\alpha-\alpha_0)cos(\delta)sin(\delta_0)-sin(\delta)cos(\delta_0)}{cos(\alpha-\alpha_0)cos(\delta)cos(\delta_0)+sin(\delta)sin(\delta_0)} \tag{11}$$

Where X,Y are our standard coordinates, α_0 is the right ascension of the center of our frame and δ_0 is the declination of the center of our frame. The code used to perform this conversion can be found in appendix 6.3.6 and 6.3.7. This can be visualized in figure 3 shown in appendix 6.2.

4 Calculations and Modeling

Now that we have our coordinates for our frames in both RA/DEC and standard coordinates, we can examine two frames taken at different times to determine the change in RA and DEC of the asteroid we chose to observe, 44 Nysa.

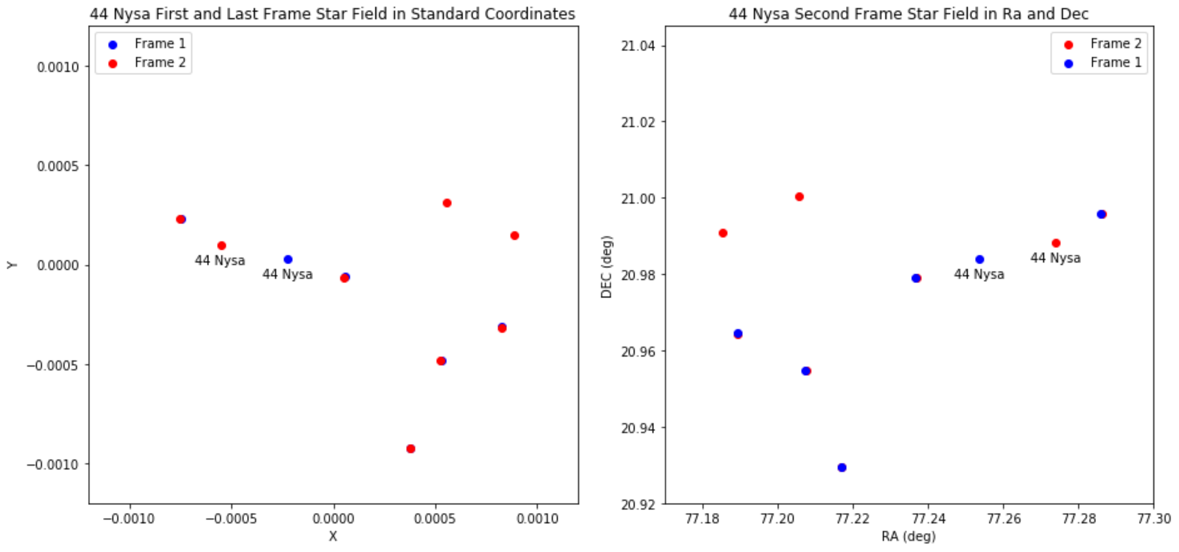


Figure 4: (Left) Shows the first and last 44 Nysa frames taken plotted in standard coordinates (Right) Shows the first and last 44 Nysa frames taken plotted in celestial coordinates. Both plots have the earlier frame (taken at 6:58 pm PST) shown in blue and the later frame (taken at 8:43 pm PST) shown in red.

The total time elapsed between the times these two frames were taken is about1 hour 45 minutes or, for simplicity, 105 ± 5 minutes, with the error being therefore discrepancy in logged time of observation and actual time of observation. The celestial coordinates of 44 Nysa in our first frame are $RA_1 = 77.2537212 \pm 0.0000017°$, $DEC_1 = 20.9839658 \pm 0.0000004°$. The celestial coordinates of 44 Nysa in our second frame are $RA_2 = 77.2738905 \pm 0.0000049°$, $DEC_2 = 20.9881811 \pm 0.0000006°$. The errors in the RA and DEC were calculated using equation (2). By finding the difference in the right ascensions and declinations, we can find the total angle the asteroid has moved by using the pythagorean theorem:

$$\theta = \sqrt{(\Delta RA)^2 + (\Delta DEC)^2} \quad (12)$$

We find the change in RA to be: $\Delta RA = 0.0201693 \pm 0.0000076°$ and the change in DEC to be: $\Delta DEC = 0.0042153 \pm 0.0000010°$. These give us a total angle change of $\theta \approx 0.0206051 \pm 0.0000092°$ with the error being given by error propagation:

$$\sigma_\theta = \theta\sqrt{\frac{(\sigma_{\Delta RA})^2}{(\Delta RA)^2} + \frac{(\sigma_{\Delta DEC})^2}{(\Delta DEC)^2}} \quad (13)$$

Converting this angle change into arcseconds using the equation $1arcsec = \frac{1°}{3600}$, we have a total angle change of 74.1784 ± 0.0331 arcseconds. Using the total change in time and total change in angle, we can calculate the asteroid's proper motion with the equation:

$$V_{proper} = \frac{\theta_{arcsec}}{t_{hours}} = \frac{74.1784 \pm 0.0331\ arcsecs}{105 \pm 5\ mins} * \frac{60mins}{1\ hour} = 42.3877 \pm 2.0186\ arcsecs/hr$$

Here we have determined the proper motion of the asteroid 44 Nysa using our observations of the asteroid.

5 Discussion

Let us visualize the proper motion of 44 Nysa as calculated in the previous section.

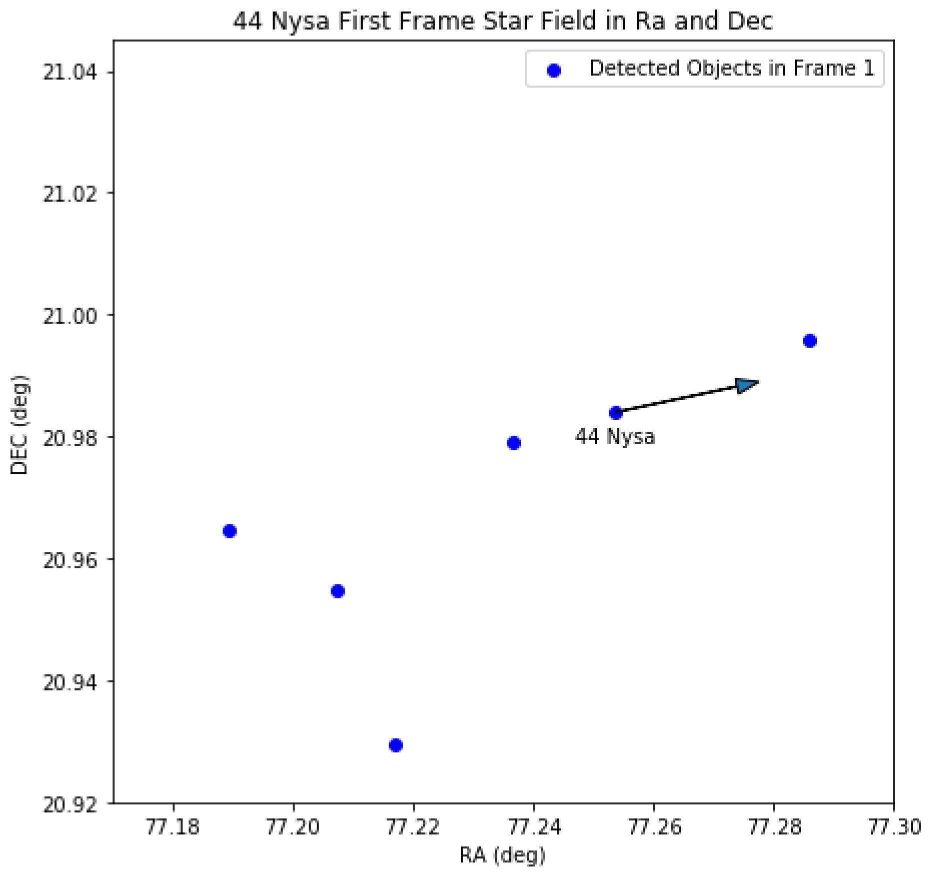
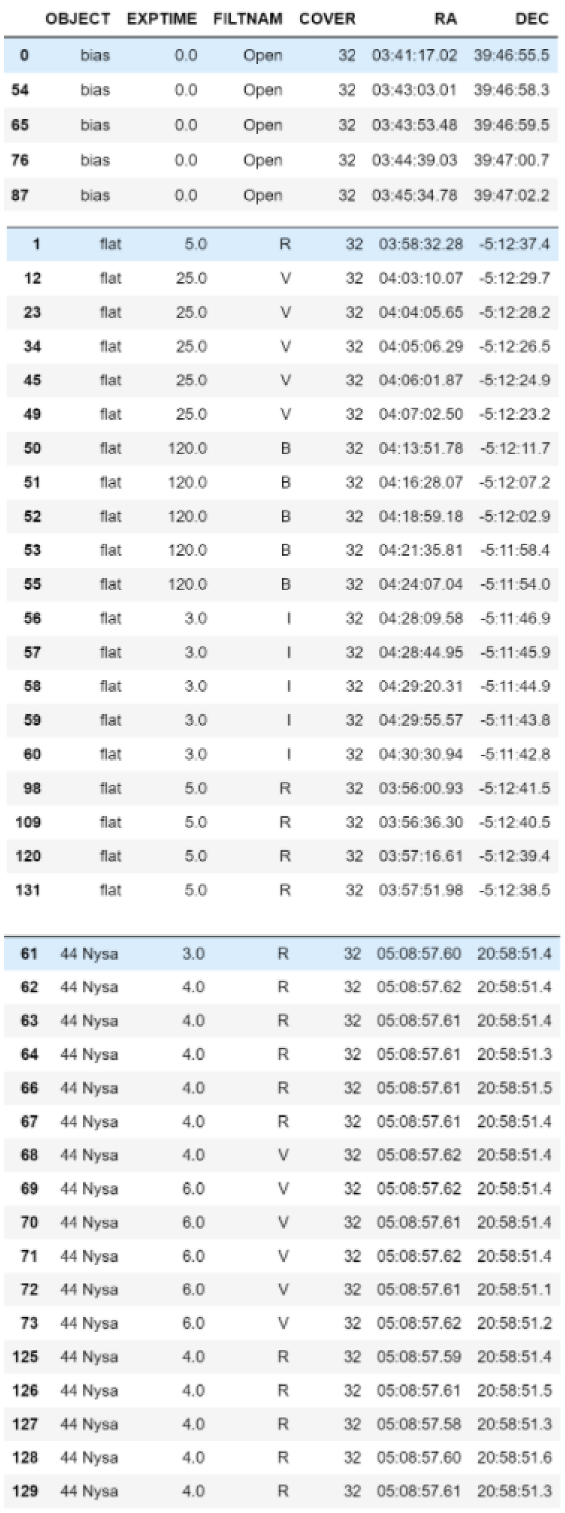


Figure 4: Plot of detected objects from the first frame analyzed with asteroid 44 Nysa labeled and its direction of proper motion shown

From figure 4, we are able to visualize the direction of motion of 44 Nysa across the celestial sphere. The largest sources of errors in our measurements come mainly from the discrepancy in our time measurements, the small error from the centroiding function, and the small error in the pixel to celestial coordinate conversion. However, we are still able to calculate the proper motion with an error of ~5% of our established measurement. More precise calculations of the proper motion could be obtained from more measurements of 44 Nysa, each in 1 hour intervals on the same night.

The proper motion of Nysa obtained from our observations does reveal several things about this asteroid. First and foremost, the proper motion being of this magnitude tells us the object we are observing must be relatively nearby as extremely distant objects' proper motion is much smaller than the proper motion obtained for Nysa. The other quantity the proper motion tells us is the relative orbit of 44 Nysa in our solar system. The angle of 44 Nysa's proper motion not being zero relative to the earth tells us that it orbits along a different axis than Earth's around the sun. Improvements to this experiment would be measuring the parallax of the asteroid by taking simultaneous measurements of the star on both sides of the continental United States. The parallax of this asteroid would allow us to calculate the distance to it and thus using our proper motion and distance calculate the transverse velocity of this stellar object. Another improvement would be to create an algorithm to automatically match the queried star locations to the observed star locations for more accurate conversion from pixel coordinates to celestial coordinates. Overall, this lab yielded a measurement of the proper motion of 44 Nysa and several algorithms useful to many different applications in astrometry.

6 Appendix
6.1 Data Files

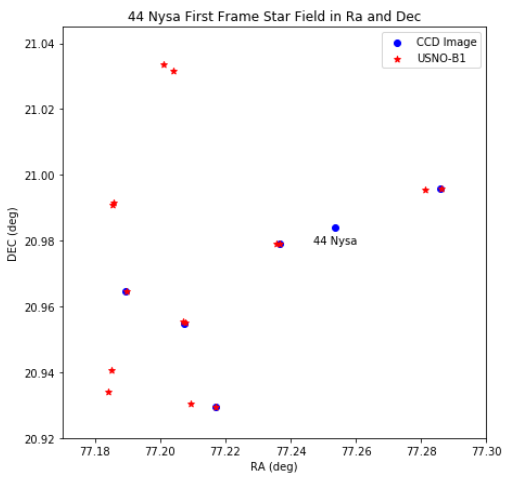


	OBJECT	EXPTIME	FILTNAM	COVER	RA	DEC
0	bias	0.0	Open	32	03:41:17.02	39:46:55.5
54	bias	0.0	Open	32	03:43:03.01	39:46:58.3
65	bias	0.0	Open	32	03:43:53.48	39:46:59.5
76	bias	0.0	Open	32	03:44:39.03	39:47:00.7
87	bias	0.0	Open	32	03:45:34.78	39:47:02.2

1	flat	5.0	R	32	03:58:32.28	-5:12:37.4
12	flat	25.0	V	32	04:03:10.07	-5:12:29.7
23	flat	25.0	V	32	04:04:05.65	-5:12:28.2
34	flat	25.0	V	32	04:05:06.29	-5:12:26.5
45	flat	25.0	V	32	04:06:01.87	-5:12:24.9
49	flat	25.0	V	32	04:07:02.50	-5:12:23.2
50	flat	120.0	B	32	04:13:51.78	-5:12:11.7
51	flat	120.0	B	32	04:16:28.07	-5:12:07.2
52	flat	120.0	B	32	04:18:59.18	-5:12:02.9
53	flat	120.0	B	32	04:21:35.81	-5:11:58.4
55	flat	120.0	B	32	04:24:07.04	-5:11:54.0
56	flat	3.0	I	32	04:28:09.58	-5:11:46.9
57	flat	3.0	I	32	04:28:44.95	-5:11:45.9
58	flat	3.0	I	32	04:29:20.31	-5:11:44.9
59	flat	3.0	I	32	04:29:55.57	-5:11:43.8
60	flat	3.0	I	32	04:30:30.94	-5:11:42.8
98	flat	5.0	R	32	03:56:00.93	-5:12:41.5
109	flat	5.0	R	32	03:56:36.30	-5:12:40.5
120	flat	5.0	R	32	03:57:16.61	-5:12:39.4
131	flat	5.0	R	32	03:57:51.98	-5:12:38.5

61	44 Nysa	3.0	R	32	05:08:57.60	20:58:51.4
62	44 Nysa	4.0	R	32	05:08:57.62	20:58:51.4
63	44 Nysa	4.0	R	32	05:08:57.61	20:58:51.4
64	44 Nysa	4.0	R	32	05:08:57.61	20:58:51.3
66	44 Nysa	4.0	R	32	05:08:57.61	20:58:51.5
67	44 Nysa	4.0	R	32	05:08:57.61	20:58:51.4
68	44 Nysa	4.0	V	32	05:08:57.62	20:58:51.4
69	44 Nysa	6.0	V	32	05:08:57.62	20:58:51.4
70	44 Nysa	6.0	V	32	05:08:57.61	20:58:51.4
71	44 Nysa	6.0	V	32	05:08:57.62	20:58:51.4
72	44 Nysa	6.0	V	32	05:08:57.61	20:58:51.1
73	44 Nysa	6.0	V	32	05:08:57.62	20:58:51.2
125	44 Nysa	4.0	R	32	05:08:57.59	20:58:51.4
126	44 Nysa	4.0	R	32	05:08:57.61	20:58:51.5
127	44 Nysa	4.0	R	32	05:08:57.58	20:58:51.3
128	44 Nysa	4.0	R	32	05:08:57.60	20:58:51.6
129	44 Nysa	4.0	R	32	05:08:57.61	20:58:51.3

6.2 Figure 3



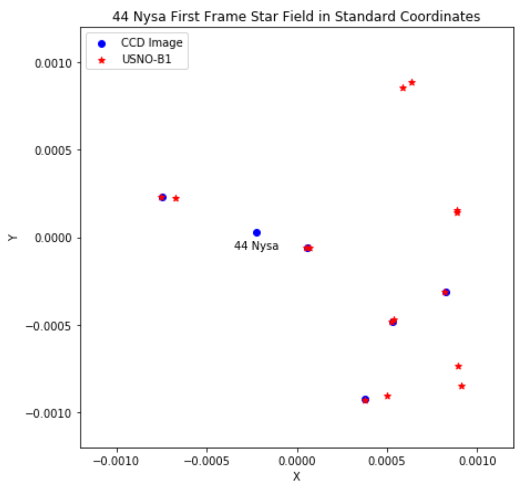


Figure 3: (left) The first 44 Nysa frame's centroid object locations converted to RA and DEC plotted against the actual star locations queried from USNO-B1. (right) The first 44 Nysa frame's centroid object locations in standard coordinates plotted against the star locations queried from USNO-B1 converted into standard coordinates

6.3 Main Coding Functions

6.3.1 Bias Subtraction and Overscan Removal Function

```python
#loads frame and removes overscan and subtracts bias
def load_frame_overscan_remove_bias_subtract(filename, overscan = 32):
    """
    Returns individual frame with Overscan removed and Bias subtracted
    Relies on: locally defined datafile names and overscan values, globally defined variables data_dir and data_files
    Averages the bias files and subtracts them from the loaded file, removes the overscan region from the x-axis
    """

    biasfiles=data_files[objects == 'bias']   #defines all bias files

    #iterates through bias files adding them
    bias = fits.getdata(data_dir+biasfiles[0])+0.0 #calls first bias file
    for i in range(1,len(biasfiles)):
        bias = bias+ fits.getdata(data_dir+biasfiles[i])+0.0
    #divides the sum of the biases to get the mean bias frame
    bias = bias/len(biasfiles)
    #loads the filename as an image and subtracts the averaged bias frames both with the overscan region removed
    image = fits.getdata(data_dir+filename)+0.0
    image = image[:,0:(len(image[0,:])-overscan)] - bias[:,0:(len(image[0,:])-overscan)]
    return image
```

6.3.2 Reduced Data Frame Function

```python
def load_reduced_science_frame(filename, filt):
    """
    Loads a Science frame specified by the FITS file name,
    performs reduction through bias subtraction and normalized
    flat field correction and returns the data
    Relies on: filename and filter (either R, B, I or V)
    returns: data, an array with columns x pixels, y pixels, intensity
    """

    #calls overscan and bias removal function for the file name
    sciframe=load_frame_overscan_remove_bias_subtract(filename)

    #creates an array for the flat field files
    flatfieldfiles = []

    #iterates through the flatfield
    for filename in data_files:
        #searches through all flatfield files
        if objects[data_files == filename][0] == 'flat':
            #searches the flatfield files with the same filter as the datafile
            if filters[data_files == filename][0] == filt:
                flatfieldfiles.append(filename) #appends all flatfield frames with the desired filter
    #iterates through all flatfield files with the same filter as the science frame
    flat = load_frame_overscan_remove_bias_subtract(flatfieldfiles[0])+0.0

    #sums the flatfield frames w/ correct filter
    for i in range(1,len(flatfieldfiles)):
        flat = flat + load_frame_overscan_remove_bias_subtract(flatfieldfiles[i])+0.0

    #divides the sum of the flatfield frames by # of flatfield files to find the mean flatfield frame
    flat=flat/len(flatfieldfiles)

    #calculates the normalized flatfield
    normFlat=flat/np.median(flat)
    normFlat = normFlat+0.0
    #multiplies the science frame by the Gain (pixel sensitivity) given by the normalized flatfield
    redsciframe=sciframe*normFlat
    return redsciframe
```

6.3.3 Star Locating Function

```python
def find_star_locs(im_data, n_size = 10, bright_count_thresh=10):
    """
    ARGUMENTS:
    ================================================================
    im_data - Reduced 2D FITS Data
    n_size  - Neighborhood size to consider for windowing (pixels)
    bright_count_thresh - Threshold for number of 'bright'pixels in
                          neighborhood to be considered a star.
                          (Proportional to size of Star blob)
    RETURNS:
    ================================================================
    [[x_positions_of_star_center, y_positions_of_star_center]]
    i.e., a list of list of x and y coordinate of star centers
    """

    list_pos=[] # sets array that will contain pixel coordinates of high value detector pixels
    brightness_thresh= np.median(im_data)*1.5 #sets base brightness threshhold to be 1.5x the median value of the file
    #sets array of indexes of all points with intensity values above the brightness threshhold
    large_ints_index=np.array(np.where(im_data > brightness_thresh)).T

     # Iterate over every potential max value pixel
    for large_ints in large_ints_index:
        #removes any indexes too close to the x edges of the image
        if large_ints[1] > np.shape(im_data)[1]-n_size or large_ints[1] < n_size:
            continue
        #removes any indexes too close to the y edges of the image
        if large_ints[0] > np.shape(im_data)[0]-n_size or large_ints[0] < n_size:
            continue
        #accounts for and removes any large intensities around the hot pixel strip at x=256
        if large_ints[1] > 252 and large_ints[1] < 258:
            continue
    # sets area to search for around large intensity values to find local maxima
        scanarea = im_data[(large_ints[0]-n_size):(large_ints[0]+n_size), (large_ints[1]-n_size):(large_ints[1]+n_size)]
        # discards any high intensity value that isn't surrounded by others (likely a hot pixel)
        bright_count = len(scanarea[scanarea > brightness_thresh])
        if bright_count < bright_count_thresh:
            continue
        # sets the local maximum in the scan range if it meets all other requirements
        if (im_data[large_ints[0], large_ints[1]] >= np.max(scanarea)):
            list_pos.append([large_ints[1], large_ints[0]])
    # sort maxima
    list_pos=sorted(list_pos,key=lambda l:l[1])
    # return

    return list_pos
```

6.3.4 Centroiding Stars from Approximate Star Location Function

```
def calc_centroids_2d(intarr, loc_list, window_max = 20):
    """
    Calculates the centroids of the image from approximate locations and the array of values

    PARAMETERS:
    ==================================================================================
    intarr - image intensity array (just the image data)
    loc_list - list of rough positions of stars
    window_max - Size of Window to consider to find max pos of each star (in pixels)

    RETURNS:
    ==================================================================================
    centroids - list of list of centroid coordinates and corresponding uncertainities
                Format: [[xc, yc, unc_xc, unc_yc]]

    """
    # Creating Position Array that is same dimension as the image we're working with
    row_dim = np.shape(intarr)[0]
    row_pos = np.array(range(row_dim))[np.newaxis]
    pos_xarr = np.tile(row_pos.T, (1 ,row_dim))
    pos_yarr = np.tile(row_pos  , (row_dim,1 ))
    pos_arr = np.dstack((pos_xarr, pos_yarr))

    centroids = []

    for i in range(0,len(loc_list)):
        scanarea = intarr[(loc_list[i][0]-window_max):(loc_list[i][0]+window_max), (loc_list[i][1]-window_max):(loc_list[i][1]+w
indow_max)]

        #defines 2d intensity arrays as a sum of columns/rows for intensity in x and y
        xInt=[]
        yInt=[]
        xPos=[]
        yPos=[]

        for j in range(0,len(scanarea[0,:])):
            xInt.append(np.sum(scanarea[j,:]))    #x 2d intensity array is the sum of each column per x pixel
            yInt.append(np.sum(scanarea[:,j]))    #y 2d int array is the sum of each row per y pixel
            xPos.append(loc_list[i][0]-window_max+j)  #finds pixel values corresponding to each x in scan area
            yPos.append(loc_list[i][1]-window_max+j)  #finds pixel values corresponding to each y in scan area

        #sets up our centroiding function by finding the sum of x*I, y*I and I
        sumxi=np.sum(np.dot(xInt,xPos))
        sumyi=np.sum(np.dot(yInt,yPos))
        sumi=np.sum(scanarea)

        #uses the calculation gone over in class to find centroids
        xcentroid=sumxi/sumi
        ycentroid=sumyi/sumi

        #uses the fullwidthhalfmaximum/signaltonoiseratio to find the uncertainty in the centroids
        xdev=0
        ydev=0
        for i in range(0,len(xPos)):
            xdev=xdev+((xPos[i]-xcentroid)**2)
            ydev=ydev+((yPos[i]-ycentroid)**2)

        #calculations for uncertainty using deviation of x and y and the sum of i
        xunc=xdev/(sumi**2)
        yunc=ydev/(sumi**2)

        centroids.append([xcentroid,ycentroid,xunc,yunc])
    return centroids
```

6.3.5 Querying from USNO-B1 Function

```python
def query_starfield_from_USNOB1(ra_cent,dec_cent, mag_lim=15, row_lim=15, fov_w='6.3m',fov_h='6.3m', year=2022):
    """
    ARGS
    - ra_cent,dec cent should be right ascension and declination of center of Aladin frame of the star field
    - mag_lim should be the desired magnitude limit of the usno-b1 query
    - row_lim should be the desired number of stars in the starfield to query
    - fov_w, fov_h should be the field of view dimensions of the frame. automatically set to '6.3m' (m is arc minutes)
    - year should be the year the observations were taken in
    Relies on:
    astroquery.simbad imported as Simbad
    astroquery.vizier imported as Vizier for vizier queries
    astropy.coordinates imported as SkyCoord to convert to standard co-ordinate objects which are used for querying
    astropy.units imported as u for unit conversions
    Returns
    ra_cat, dec_cat  which are queried right ascension and declination of stars in the star field
    also creates a plot of the queried star field for visualization
    """
    ra_center = ra_cent    #'05:08:57.6' for 44 nysa
    dec_center = dec_cent    #'+20:58:51.4'for 44 nysa

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

    magnitude_limit = mag_lim    #sets upper magnitude limit

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

    #tells vizier to give the top 15 star results with these conditions in the given field.
    vizier.ROW_LIMIT = row_lim

    # "catalog = "USNO-B1" tells VizieR which catalogue we want. This could be a list of multiple catalogues.
    fov_width  = fov_w
    fov_height = fov_h

    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())
    # Extract required data from obtained query results
    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"])

    # Convert result into form we want

    epoch = year      #relates the year to find the correct RA and DEC of stars in the field
    # this should be a fractional number, but most stars do not move fast enough to make the difference matter.

    # 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
    # gives the FOV with the starfield present
    plt.plot(ra_cat, dec_cat, 'r.')
    plt.xlabel('Right Ascencion (Degrees)')
    plt.ylabel('Declination (Degrees)')
    plt.title('Star Field in designated area')

    for i in range(0,len(ra_cat)):
        plt.annotate(i+1,(ra_cat[i],dec_cat[i]),textcoords="offset points", xytext=(0,10), ha='center')
    return ra_cat,dec_cat
```

6.3.6 Converting Centroids from Pixels to RA/DEC Function

```
## Converting from pixels to RA and DEC
def convert_to_ra_dec(starpairs , centroids, ra_cat, dec_cat):
    """
    ARGS
    - starpairs var should be a list of adjacent observed and queried stars
    - centroids should be the list of centroids
    Relies on ra_cat and dec_cat variables to define actual star RA & DEC
    Relies on a locally defined centroid list to be entered as var centroids
    Relies on locally defined star pairs from the

    Returns centroid_in_ra_dec which will be all centroid values converted from pixels to ra and dec
    """


    f_p = 190020 #units of pixels/radian given by lab handout
    f_p = f_p*np.pi/180  #converstion of f_p to pixels/degree

    #defines the pixel and ra/dec matrixes
    x_pix = []
    y_pix = []
    ra=[]
    dec = []
    for i in range(0,len(starpairs)):
        x_pix.append(centroids[starpairs[i][0]-1][0])
        y_pix.append(centroids[starpairs[i][0]-1][1])
        ra.append(ra_cat[starpairs[i][1]-1])
        dec.append(dec_cat[starpairs[i][1]-1])

    #matrix chi suqared fit for x coordinates

    ones = np.ones(len(x_pix))
    #creates a matrix of [(f/p)*ra (f/p)*dec 1] for all ra and dec pairs listed in the star pairs variable
    MAT = np.dot(f_p,ra), np.dot(f_p,dec), ones
    #transposes the matrix to be of the form desired for the matrix multiplication
    B=np.transpose(MAT)
    BT=np.transpose(B)
    #inverse of transposed B matrix dot producted with the B matrix
    BT_B=np.linalg.inv(np.dot(BT,B))
    # defines our first set of constants from this fit for conversion of x pixels
    a_11, a_12, x_0 = np.dot(np.dot(BT_B,BT),x_pix)
    # defines our second set of constants from this fit for conversion of y pixels
    a_21, a_22, y_0 = np.dot(np.dot(BT_B,BT),y_pix)

    #use the variables found above to construct a transformation matrix T to convert from pixels to RA/Dec and vice versa
    T=[[(f_p*a_11), (f_p*a_12), x_0],[(f_p*a_21), (f_p*a_22), y_0], [0, 0, 1]]

    #ra_dec = np.array([ra, dec, ones])
    #defines our array of pixels with the form [x,y,1]
    pixels = []
    for i in range(0,len(centroids)):
        pixels.append([centroids[i][0],centroids[i][1],1])

    #uses the matrix multiplication X=x*T^-1 to convert our coordinates from pixels to RA and DEC from the constants calculated
    centroids_in_ra_dec = []
    for i in range(0,len(centroids)):
        centroids_in_ra_dec.append(np.dot(np.linalg.inv(T),pixels[i]))

    #print(centroids_in_ra_dec)
    return centroids_in_ra_dec
```

6.3.7 Converting Centroids in RA/DEC to Standard Coordinates

```python
## converting to standard coordinates
def project_coordinate(ra_centroid,dec_centroid, center_ra, center_dec, axis):
    """
    Args
    - ra_centroids should be an array of ra coordinates of a centrioded object
    - dec_centroids should be an array of dec coordinates of a centroided object
    - center_ra,center_dec should be the right ascension and declination of the center of the frame
    - axis should be either x or y and determines if a standard X or Y coordinate should  be calculated
    Returns
    - X, which is the either standard X or Y coordinate depending on which axis was entered
    """
    if axis == 'x' or axis == 'X':
        X = -1*(np.cos(dec_centroid)*np.sin(ra_centroid-center_ra)/(np.cos(center_dec)*np.cos(dec_centroid)*np.cos(ra_centroid-c
enter_ra) + np.sin(dec_centroid)*np.sin(center_dec)))
    if axis == 'y' or axis == 'Y':
        X = -1*(np.sin(center_dec)*np.cos(dec_centroid)*np.cos(ra_centroid-center_ra) - np.cos(center_dec)*np.sin(dec_centroi
d))/(np.cos(center_dec)*np.cos(dec_centroid)*np.cos(ra_centroid-center_ra)+ np.sin(dec_centroid)*np.sin(center_dec))

    return X

def convert_from_celestial_to_standard(ra_centroids,dec_centroids, center_ra, center_dec):
    """
    ARGS
    - ra_centroids should be an array of all ra coordinates of the centrioded objects
    - dec_centroids should be an array of all dec coordinates of the centroided objects
    - center_ra,center_dec should be the right ascension and declination of the center of the frame
    Converts centroids coordiantes from degrees to radians and calls the project_coordinates
    function to find the standard X and Y coordinates of the objects
    Returns
    standardX - array of all standard X values of the centroids
    standardY - array of all standard Y values of the centroids
    """
    ra_cents = np.deg2rad(ra_centroids)
    dec_cents = np.deg2rad(dec_centroids)
    centerRA = np.deg2rad(center_ra)
    centerDEC = np.deg2rad(center_dec)

    standardX=[]
    standardY=[]
    for i in range(0,len(ra_centroids)):
        standardX.append(project_coordinate(ra_cents[i],dec_cents[i], centerRA, centerDEC, 'X'))
        standardY.append(project_coordinate(ra_cents[i],dec_cents[i], centerRA, centerDEC, 'Y'))

    return standardX, standardY
```