Antony Sikorski
Phys 164
Professor Konopacky
12/12/2020

# Physics 164 Lab #3: Astrometry from CCD Images

Authors: Antony Sikorski, Lucas Scheiblich
Contact: asikorsk@ucsd.edu, lscheibl@ucsd.edu
12, December, 2020

## Abstract

Astrometry is a branch of astronomy that deals with the precise position and motion of celestial objects. Such information about celestial bodies is often critical in deepening our understanding of their properties, along with determining how they may affect us. In the case of asteroids, these calculations allow us to find the probability of an asteroid colliding with our planet, and ending our civilization as we know it. In this lab, we focus on the specific measurement of proper motion, which is the apparent movement of an object across the celestial sphere, and perpendicular to the observer's sight. The objective was to calculate the proper motion of Parthenope, a large, bright, main belt asteroid, using the Nickel 1-m telescope at Lick Observatory. Two images were taken at a time of two hours apart in order to observe the motion of the asteroid. The images were then reduced, and the positions and centroids of the asteroid and the surrounding stars were calculated. The USNO catalog was used as a reference for the positional measurements of the stars. After this, the images could be corrected, and the initial and final positions of the asteroid were used in order to calculate it's proper motion. As a result, we observe Parthenope to have a proper motion of approximately -1.08 ± 0.02 arcseconds per hour for the right ascension, and -1.21 ± 0.02 arcseconds per hour for the declination. While our results only find the proper motion, this presents

the opportunity for further calculations such as angular velocity and orbit, which will give a more complete understanding of Parthenope's motion.

## Section 1: Introduction

In roughly 400 BC, after much observation, Eudoxus of Cnidus and Callippus of Cyzicus developed the first three dimensional models to explain the apparent motion of the planets and stars they saw in the night sky. This was the first semblance of astrometry, a major component of astronomy that focuses on specific calculation regarding the position and motion of celestial bodies. Since then, astrometry has come a long way, and since modern technology allows us to make extremely accurate observations of celestial objects, we now have the power to calculate the majority, if not all, of their kinematic properties. One of the most basic properties of a moving object is it's proper motion. The proper motion of an object, as previously stated, is basically it's velocity perpendicular to our observation. In this lab, we are tasked with calculating the proper motion of the asteroid, Parthenope.

Although it is a main belt asteroid, Parthenope is classified as a minor planet due to its size. It has a mass of 6 ✕ 10^18 kg, and a density of about 3.3 grams/cm^3. Luckily for our observations, Parthenope is a very bright asteroid, with an apparent magnitude of 8.8, and orbits the Sun in about 4 years. Parthenope is also an S-type asteroid, meaning it is within the second most common group of asteroid types. S-type asteroids are silicaceous, and also have rich iron and magnesium content, and are most likely to be found in the inner part of the main asteroid belt, although Parthenope is not.

Shown below in **Figure 1** is a three dimensional model of Parthenope, which demonstrates both it's apparent color and it's shape.

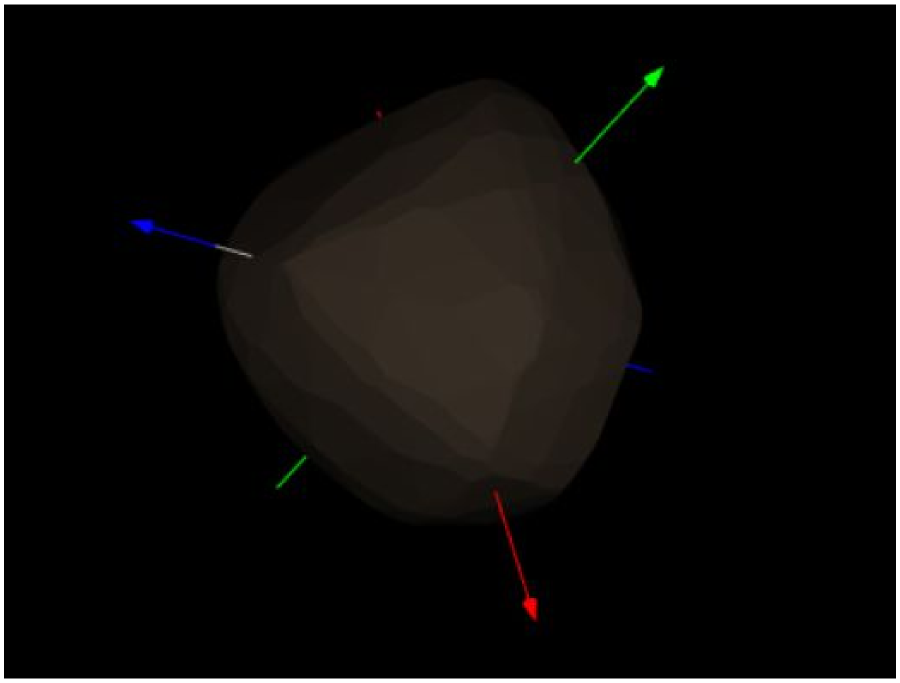
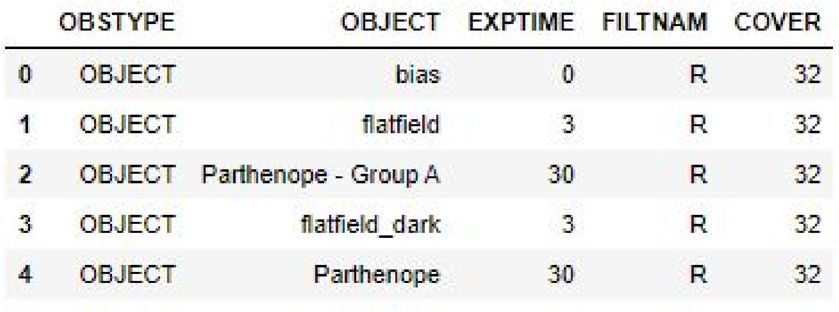


**Figure 1:** Three dimensional model of the asteroid, Parthenope, which shows both it's apparent, brown color, and it's rough geometric shape. https://3d-asteroids.space/asteroids/11-Parthenope

## Section 2: Observations and Data

In order to observe the proper motion of an asteroid, it is necessary to take two images. Due to the low brightness of this asteroid, it was necessary to observe it on a clear night when it was most visible, which happened to be November 23rd 2020. We took our images two hours apart, the first at 2:39 UT (6:39 PST), and the second at 4:48 UT (8:49 PST). In order to get the most precise measurement, it was necessary to perform an observational planning routine, where multiple factors were accounted for.

The rise time, transit time, and coordinates of the asteroid were initially searched for to determine whether it was viable to observe. After that, the magnitude, necessary filter, and exposure times in order to get the best image were calculated. No data is available for this process will be given due to the fact that my lab group originally planned to observe another asteroid, Eurynome, but the telescope was unavailable due to high wind conditions, so we chose to analyze the data from Parthenope. Each image from Parthenope is taken twice due to the fact that the exposure time had to be set to higher than initially anticipated, which was 30 seconds. In addition to these images, both bias and flatfield images were taken for the purposes of data reduction, which will be explained in detail in **Section 3**. The images were then saved as FITS files so we could analyze them in Python via JupyterHub.

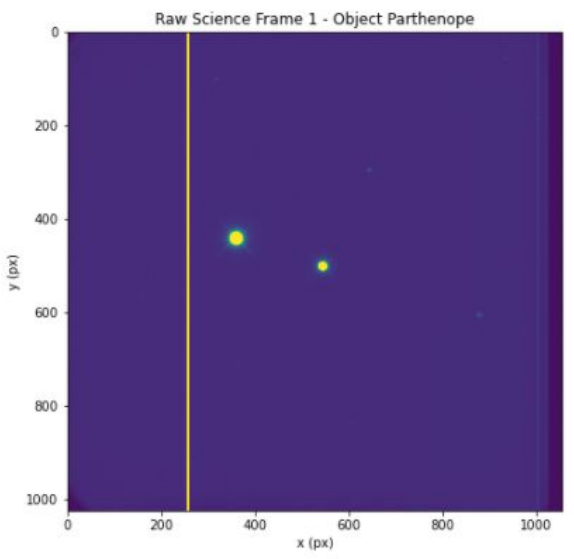


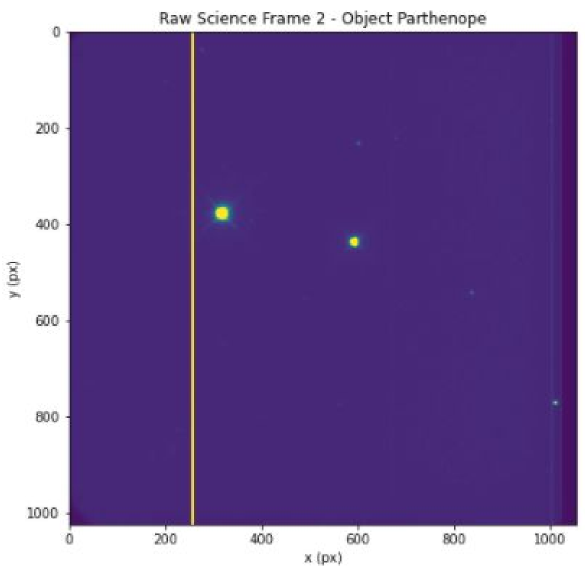
| | OBSTYPE | OBJECT | EXPTIME | FILTNAM | COVER |
|---|---|---|---|---|---|
| 0 | OBJECT | bias | 0 | R | 32 |
| 1 | OBJECT | flatfield | 3 | R | 32 |
| 2 | OBJECT | Parthenope - Group A | 30 | R | 32 |
| 3 | OBJECT | flatfield_dark | 3 | R | 32 |
| 4 | OBJECT | Parthenope | 30 | R | 32 |

**Table 1:** FITS files used in the experiment with all relevant headers displayed in the table above. The specific fits file numbers are 4, 5, 24, 40, and 90, respectively. **Function 2** creates this display.

## Section 3: Data reduction and Methods

In order to both get clear images, and the most accurate results possible, the data had to be reduced. Initially, the data was 2x2 binned in order to reduce the file size and to read out some of the noise. Next the overscan region of the data had to be removed, which accounted for the last 32 columns of the image. Bias frames are frames with zero exposure that account for the high probability that there was still some form of information recorded without ever exposing your instrument, and so they are subtracted from both the science and flat frames that are taken. The flat frame is normalized by it's median value, and the science frame is divided by the normalized flat frame to create the final product. This provides a significantly cleaner image, which allows for more faint objects, such as Parthenope, to be detected. Below in **Figure 2**, a comparison between the both raw and reduced science frames is shown. The figures are reduced using **Function SOMETHING**.





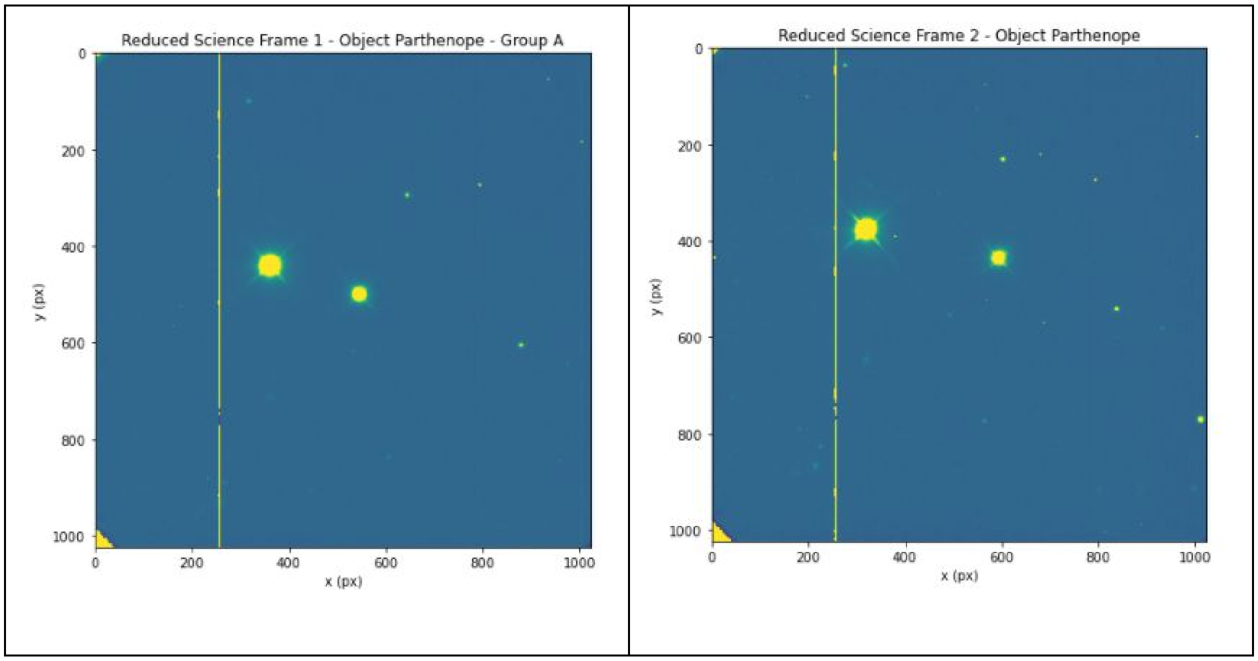


**Figure 2:** Raw (top left and right) and reduced (bottom left and right) science frames for both observation times are shown above.

It is evident that the reduction was effective, as more stars appear in the reduced images, the overscan region is truncated, and the original stars that could be viewed in the raw images are much brighter. The reduced frames are now clear enough to begin performing the necessary calculations that lead to finding the proper motion of Parthenope.

## Section 4: Calculations and Modeling

In order to begin the calculations, the first step was to locate the stars and the asteroid in each of our reduced science frames. **Function 5** is used to find stars based on their intensity, shape, and pixel position within the frame. Below, **Figure 3** displays the stars by placing a circle around each location and numbering them.

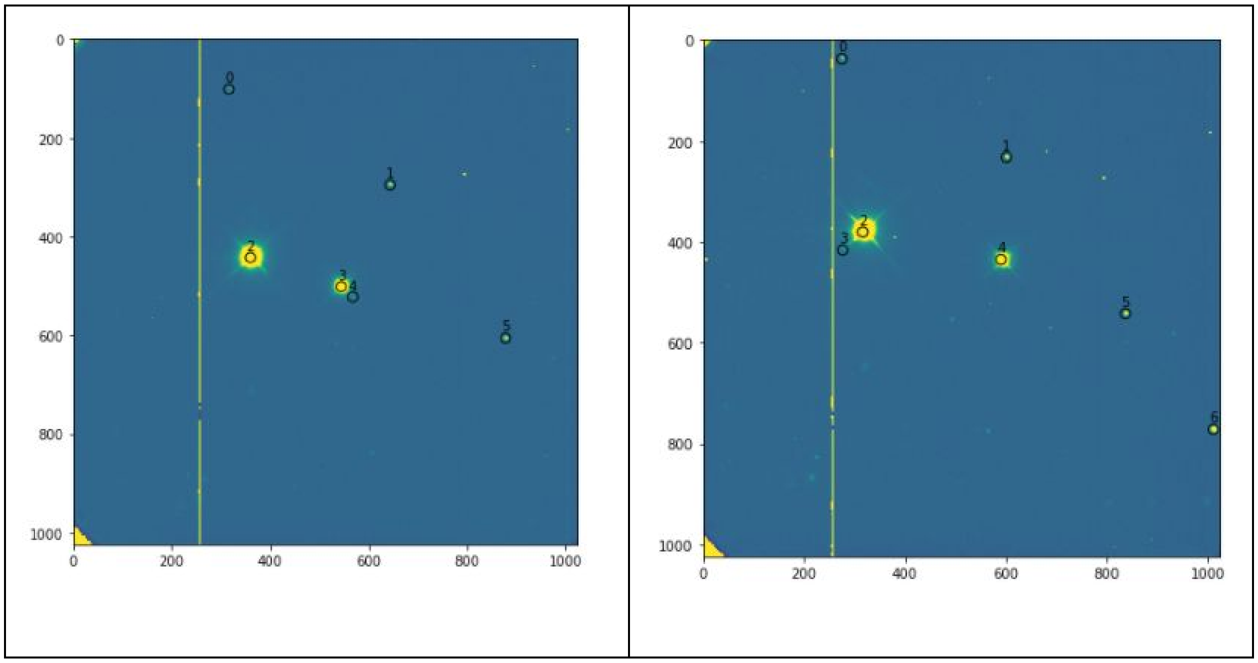


**Figure 3:** The star locations are shown in both images, and are represented by a circle being placed around each star, along with a number above them in order to keep track of the positions. Our asteroid appears as number 3 in the first image, and number 4 of the second image.

As demonstrated above, the asteroid is clearly found in the above image, where it is first labeled as number 4, but then labeled as number 3 in the second image. This is a consequence of the code, and is taken into account in our future calculations to avoid a trivial labeling error. With the positions found, the ability to calculate the centroids of the stars is much easier. This is because the objects that we are finding the centroids of have already been filtered via some initial conditions, such as size and shape, meaning they are all roughly circular. The centroids of the star are found using **Function 6**, which uses the average weighted mean of pixel intensity to most accurately find the center of our objects.

Now that the centroids have been found, it is time to match our centroids with queried coordinates from the USNO databases. We contact the database and search for stars with a low magnitude to avoid a reference frame with an abundance of stars, which would make matching the data considerably more difficult. The task still proved to be quite difficult, and despite rigorous attempts, the most lined up image produced is shown below in **Figure 4**.

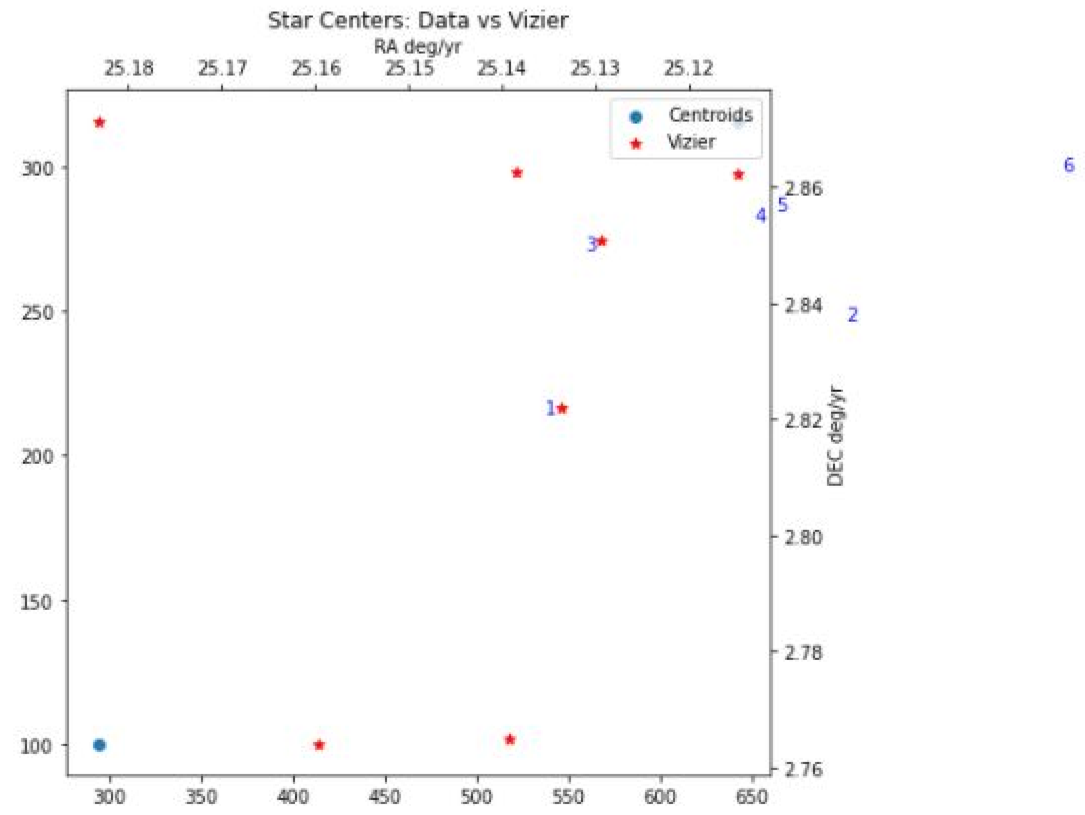


**Figure 4:** Our star data aligned with data queried via Vizier, with a Right Ascension: 01:40:35 and Declination: 02:49:00. The FOV of this frame is 7x7 arcminutes. Alignment looks relatively unsuccessful, but proves to be quite accurate due to good plate constant results.

The matching, although painful, is required in order to correct for shear and rotational distortion in the image. With the assumption of an ideal camera, the coordinates could be easily converted, with the only information needed being the focal length of the camera and the pixel size. We cannot assume an ideal camera, so we must calculate the plate constants, which will allow for the final correction, and lead to the calculation of proper motion. First the data must be converted from right ascension and declination on the celestial sphere to X and Y pixel positions using **Equation 1**.

$$X = -\frac{\cos\delta \sin(\alpha - \alpha_o)}{\cos\delta_o \cos\delta \cos(\alpha - \alpha_o) + \sin\delta \sin\delta_o}$$

$$Y = -\frac{\sin\delta_o \cos\delta \cos(\alpha - \alpha_o) - \cos\delta_o \sin\delta}{\cos\delta_o \cos\delta \cos(\alpha - \alpha_o) + \sin\delta \sin\delta_o}$$

**Equation 1:** Equations for RA and DEC to X,Y conversion with δ being the declination, and ɑ being the right ascension.

Now, we use matrix functions to correct for the shear and rotational distortion. For this, we will use the general affine transformation, with terms for shear, scale, and orientation of the offset. The calculation for the six unknown plate constant variables is shown below in **Equation 2**.

| | | |
|---|---|---|
| $\mathbf{x} = \mathbf{TX}$ <br> $\mathbf{T} = \begin{bmatrix} (f/p)a_{11} & (f/p)a_{12} & x_o \\ (f/p)a_{21} & (f/p)a_{22} & y_o \\ 0 & 0 & 1 \end{bmatrix}$ | $\mathbf{c} = \begin{bmatrix} a_{11} & a_{12} & x_o \end{bmatrix}$ <br> $\mathbf{a} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$ <br> $\mathbf{B} = \begin{bmatrix} (f/p)X_1 & (f/p)Y_1 & 1 \\ (f/p)X_2 & (f/p)Y_2 & 1 \\ \vdots & \vdots & \vdots \\ (f/p)X_N & (f/p)Y_N & 1 \end{bmatrix}$ <br> $\mathbf{a} = \mathbf{Bc}$ | $\mathbf{c} = (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{a}$ |

**Equation 2:** The general affine transformation (left), a, B, and c matrix equations (middle), final relation which allows for the solution of $a_{ij}$, $x_0$, and $y_0$ .

After finding the plate constants, we use the distance formula to find the distance between the actual coordinates of the asteroid in the first and second image. Dividing this by the time between the images, which was 2 hours, results in a calculation for the change in both right ascension and declination per hour. The maximum error is calculated using **Function 7**, which uses the mean error of both images' centroid calculation as an upper bound.

The final calculated values are:

| Right Ascension Velocity (arcsec/hr) | Declination Velocity (arcsec/hr) |
|---|---|
| $-1.1 \pm 0.02$ | $-1.2 \pm 0.02$ |

## Section 5: Discussion

In order to see whether our calculations roughly match up, it is important to do a quick visual assessment. Both the RA and DEC velocities that we have calculated come out to be negative, meaning the asteroid should be moving towards the bottom right of the screen, which it does appear to do after glancing from the first image to the second one. Below, **Figure 5** shows an approximate velocity vector based on visual inspection, which matches up with our calculated values.

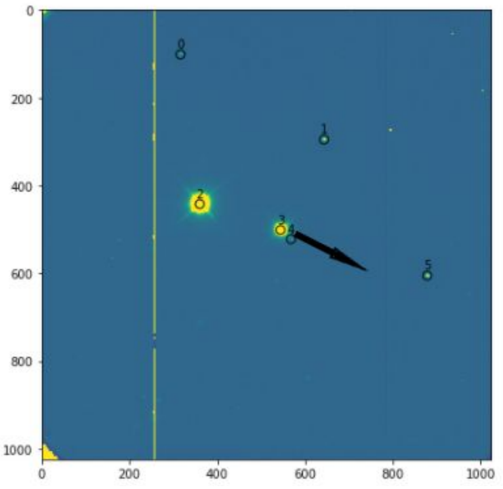


**Figure 5:** Approximate velocity vector added to the asteroid to show it's path.

In addition to the visual observation, the observed velocities are similar to what is expected for an asteroid being observed at this distance, so while there are many sources of error that must be discussed, and improved on, this was a generally accurate calculation.

As stated above, there are numerous sources of error. To begin, there is a very bright star right next to the asteroid, which influences the mean brightness of our data, and could have some effect on our centroid and star locator functions. Next, the images appear to be shifted, and this influences the calculations, so it would’ve been more helpful to take them from the exact same telescope position both times. Additionally, many more images should’ve been taken, and this calculation should’ve been redone multiple times in order to gain an actually accurate sense of the asteroid’s motion. This would’ve also eliminated multiple other sources of error in the data, which are automatically present due to the fact that we are only working with two instances. The motion that was calculated from these two images could in fact be incorrect, and taking the average motion, or observing the change in the velocities over multiple frames would produce the best result.

Despite these potential sources of error, the experiment proves to be an overall effective exercise into performing the basic calculation of proper motion of Parthenope. This already allows us to predict where this asteroid could later be observed in the night sky on a given date. The possibility of future experiments and calculations is vast, due to the fact that proper motion is not a very effective estimator of a celestial bodies’ motion alone. The angular velocity and orbit could be measured to gain a more holistic understanding of the kinematic properties of Parthenope. Additionally it would be helpful to calculate the probability of Parthenope colliding with Earth, and placing it on the Torino scale, which would save the Air Force some time.

Antony Sikorski
Phys 164
Professor Konopacky
12/12/2020

## Section 6: Appendix

All of the in house functions and figures were created by myself and my lab group in a collaborative effort. Our lab group consists of myself, Antony Sikorski, and my lab partner Lucas Scheiblich. No particular function was dominated by one of the members of the lab group, because each one was thoroughly verified and improved by the remaining group members. Code was also sampled and used from most of the Discussions and Coding Exercises, which was permitted by Caleb Cohan.

### In House Functions (All written in Python):

**Function 1: Modified Sine and Cosine Functions (for coordinate calculations)**

```
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)
```

**Function 2: FITS File Header Function**

```
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
```

**Function 3: Flat Reduction Function**

```
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
```

**Function 4: Science Image Reduction Function**

```
def load_reduced_science_frame(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
```

**Function 5: Star Location Finder**

This function finds the location of the stars based on their pixel position, brightness, and shape.

```
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
```

**Function 6: Centroid Finder**

This function returns the x and y coordinates of the centroids of our located stars.

```
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
```

**Function 7: Centroid Error Function**

This function finds the error on each individual centroid, and additionally returns the mean error for the group.

```
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
```