AST325H1F Lab 3: Astrometry

Parampreet Singh
parampreet.singh@mail.utoronto.ca
Lab Group G

December 5th, 2019

Abstract

In this lab report, Charged Coupled Device (CCD) images from the Dunlap Institute Telescope (DIT) were used to measure the positions of stars relative to the celestial coordinate system to determine proper motion of the asteroid 26-Proserpina. The ambition for this lab was to get familiar with the astronomical skills needed to track the positions of some minor planets (asteroids) over the course of three to four weeks. This was done through first applying systematic corrections to provided data from the DIT CCD. Through the provided data, analysis such as determining star positions, interrogating the USNO star catalogue, and determining matches between these data sets were performed. Then, the position and proper motion of 26-Proserpina were measured by generating and contrasting various plots. The plots outlining the motion of 26-Proserpina will be presented in this lab report. The magnitude of proper motion for 26-Proserpina was determined to be 18.6 [arcsec/hr] with $\pm$ 2.5 [arcsec/hr] and $\pm$ 1.3 [arcsec/hr] uncertainties in the right ascension and declination directions respectively. To produce uncertainties, two types of errors were focused on. One being a distance error and the other being the pixel error. These errors alongside other sources of errors will be discussed and the reduction of these errors will also be explored.

1 Introduction

Astrometry is a branch of astronomy which concerns the precise measurements and movements of stars and other celestial bodies. For astrophysicists, Astrometry is a very essential and fundamental tool for producing astronomical measurements. Astrometry is used in a variety of ways, some of which include determining distances to nearby stars using parallax, investigating proper motions of stars, calculating orbits of solar system objects, binary stars, exoplanets, stellar populations, and observing astrometric microlensing [3].

A regular example of an Astrometrical method is to determine the proper motion of close objects in relation to the movement of that object compared to background stars. These background stars are located much further away giving the approximation that they are fixed. For this lab, this regular method of Astrometry was used in order to determine the proper motion and position of an asteroid known as 26-Proserpina. To determine these quantities, measurements taken from the Dunlap Institute Telescope (DIT) were compared to the United States Naval Observatory (USNO) star catalogue to determine the movement of 26-Proserpina over a well-defined time period. Here, 26-Proserpina is treated as the close object and stars from the USNO star catalogue are treated as the much more distant objects. Through observations of a single patch of sky conduced by the DIT over many days, it was possible to trace out the movement of 26-Proserpina when one is certain of the star positions of each observation day (by consulting the USNO star catalogue).

In this experiment, we attempt to determine the position and proper motion of 26-Proserpina by conducting a series of detailed steps outline by a lab handout which can be found on the AST325 course website [2]. This is mainly done through first reducing the DIT data for each observation day and then determining the star positions for these data sets through the center of light method as described in Section 3. The following star positions are then compared to the USNO star catalogue to determine any matches between the two data sets. After this, the detection of 26-Proserpina is very clear compared to matched star positions, and the

proper motion and position can be determined. Through this process, methods such as standard coordinate transformations and pixel conversions (Section 4.1) are heavily used. To conduct a proper error propagation of all processes, the method of determining plate constants using least squares is employed (Section 4.2). This gives detailed information between the discrepancies of the observed and predicted transformation mappings in terms of pixel error.

The division of labour this lab was split evenly between the Author (Parampreet Singh) and Lab Group G members Nicholas Clark, Juan Pablo Alfonzo, and Lucas Louwerse. The methods and Python programming codes used to obtain and analyze data were determined as a partnership between all four group members. Individual members had their own approach to compile various data files and produced their own unique results, plots, and fit calculations. These were compared between group members to ensure the accuracy of the experiment.

2 Observations and Data

For this lab, all raw data was fully provided from two major sources. The first source of data was the DIT which is a a 50-cm robotic telescope dedicated to the search for gamma-ray bursts. The telescope was housed in a dome located at New Mexico Skies on the Mt. Joy site near Mayhill, NM. The telescope is equipped with a 4096 x 4096 pixel CCD array. The resultant field of view is approximately 36 x 36 arc minutes. More details of this specific telescope can be found on the AST325 Lab 3 Handout [1]. Data from the DIT presented itself in terms of a file format known as `FITS` or `FTS`. This file format is a combination of two types of stored data. The first being an image, and the second being a 2048 x 2048 array which describes the intensity of a source at specific x and y pixel positions of that image. In total, four separate observations were conducted by the DIT on four different days. For each day, three raw data files for a patch of sky, five flat files (to correct for uneven light dispersion over the telescope lens), and fifteen dark files (to correct for electromagnetic radiation being given of by the CCD sensor itself) were provided. A log of the observations made from the DIT are provided in Table 1.

Table 1: DIT Observations.

Beginning of Data			
Data File	x Pixel Number	y Pixel Number	Measured CCD Signal
Raw	889	1034	65535
Flat	889	1034	20395
Dark	889	1034	971
End of Data			

Shown in Table 1 are sample data cells corresponding to one Raw file (`26Prosperina-S001-R001-C001-r.fts`), one Flat file (`AutoFlat-Dusk-r-Bin2-001.fts`), and one Dark file (`Dark-S001-R001-C001-B2.fts`) from day one observations conducted by the DIT. For a given x and y pixel, a corresponding CCD measurement is provided. Note that a single 2048 x 2048 array would produce 4,194,304 individual cells. Therefore, one cell of data for each data file has been provided (for briefness) as a reference. In general, all cells contain a similar format of stored data and it is sufficient to present only some cells of data because of this.

The second source of data was the USNO star catalogue which is an all-sky catalog that presents positions, proper motions, magnitudes in various optical passbands, and star/galaxy estimators for 1,042,618,261 objects derived from 3,643,201,733 separate observations. More details of this specific catalogue can be found on the AST325 Lab 3 Handout [1]. Data from the USNO catalogue was displayed as a two-dimensional plot specified by a Right Ascension [Degrees] and a Declination [Degrees] for a certain patch of sky. To access the catalogue of a certain part of the sky, the user must input four parameters. These parameters are the name of the catalogue (in this case, USNO), the Right Ascension and Declination to specify where the plot will be centered, and a Field of View (in this case it is a 36.4 arc minute square as provided in the lab handout). A sample plot is shown in Figure 1.

Shown in Figure 1 are a few catalogued stars from the USNO database. The four specified parameters include USNO(81.0333 [RA], 26.9808 [DEC], 36.4 FOV [36.4 x 36.4 arc minute square]). This correlates to

a USNO catalogue of stars centered at 81.0333 Right Ascension, 26.9808 Declination, and a 36.4 FOV.

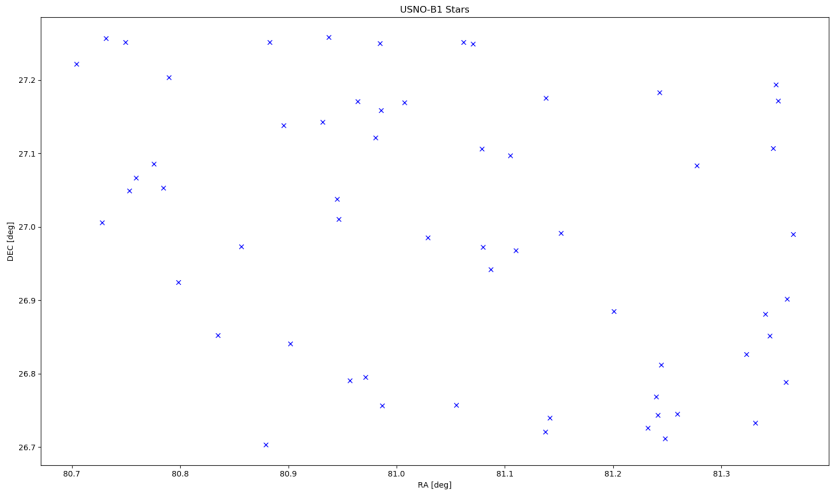

Figure 1: Catalogued USNO Stars

Both sources of data come from very reliable institutions. Therefore, it can be assumed, for the most part, that any anomalies or systematic errors associated with the provided data are minimized. However, it is worth noting that all data for this experiment was fully provided. This means that it is uncertain whether or not there could be any sort of errors attached to each source of data. As a result, the accuracy and precision of the data sets is very questionable. Since there is no way to counter these errors as someone who's main task is to process the data, this is just something that we have to keep in mind moving forward through the lab.

3 Data Reduction and Methods

For this lab, the provided data was analyzed using numerous Python coding packages. These include `matplotlib`, `numpy`, `astropy`, `astropy.io`, and `urllib.request`. This was combined with many algorithms and functions, created by the Author, which heavily incorporate these packages for the purpose of processing the given data and providing the required analysis. For this lab in particular, the required tasks were to perform a reduction of data following instructions which were given through a lab handout [1]. What is unique is how each group member went about tackling the method for reducing and processing their data. The main goal of this experiment was to determine the position and proper motion of 26-Prosperina. This includes performing tasks such as applying systematic corrections to provided data, generating multiple plots to visualize star position data and the movement of 26-Prosperina, converting pixel positions to standard coordinates and vice-versa, and also computing multiple error factors for these transformation mappings.

The first step in the reduction of data focused on applying systematic corrections to given `FITS` data files from the DIT. To do this, the Python coding packages `astropy` and `astropy.io` were implemented in order to import data directly from the DIT into a Python code. A direct example of this can be found under Appendix A. As mentioned in Section 2, each observation day included three raw data files, five flat files, and fifteen dark files. After correctly importing all necessary `FITS` data files (twenty-three in total), the median values for each type of data file was computed. This is because with multiple observations, it is best to take a middle value for each type of data file to ensure all data is on equal footing. This also provides for a better estimate of the true conditions for each set of data files. The Python code used to calculate the medians of data files can be found under Appendix A. The method of determining medians is defined in the following way where $\tilde{x}$ represents the median value, and $a_1, a_2, ..., a_n$ represent arbitrary data points ordered from least to greatest:

$$\tilde{x} = a_{(n+1)/2} \tag{1}$$

The second step in the reduction of data focused applying systematic corrections to raw data files. Using Eq. 1, it was possible to reduce twenty-three total files to three total files (one for each of the raw, flat, and dark files). From here, a simple Python correction method, as highlighted in Appendix A, was used. This method was derived from the following equation where C represents a corrected data set, R represents a median raw data set, D represents a median dark data set, and F represents a median flat data set:

$$C = \frac{R - D}{F - D} \tag{2}$$

With a corrected data set now available, the last step in the reduction was to determine the positions of stars in this data set. Since star positions are more accurately described by their centroids, the center of light method was employed in order to determine this quantity. The center of light method is a type of weighted average for the intensity with relation to pixel positions. Since C contains 2-dimensions of pixels (x and y), the center of light method must be calculated both the x and y directions. The python code for this method can be found under Appendix B. The center of light method is described by the following equation where $\bar{x}$ and $\bar{y}$ represent the x and y centroid position, x_i and y_i represent the i^{th} pixel position in the x and y directions respectfully, I_i represents the intensity at the i^{th} pixel position, and i represents the indexing range of a star:

$$\bar{x} = \frac{\sum_i x_i I_i}{\sum_i I_i} \qquad \bar{y} = \frac{\sum_i y_i I_i}{\sum_i I_i} \tag{3}$$

From the methods highlighted through Eq. 1-3, one is left with a corrected data set in which star positions have been found and recorded, as shown in Figure 2. Note that the entire reduction process was done for all observation days in a similar manner.

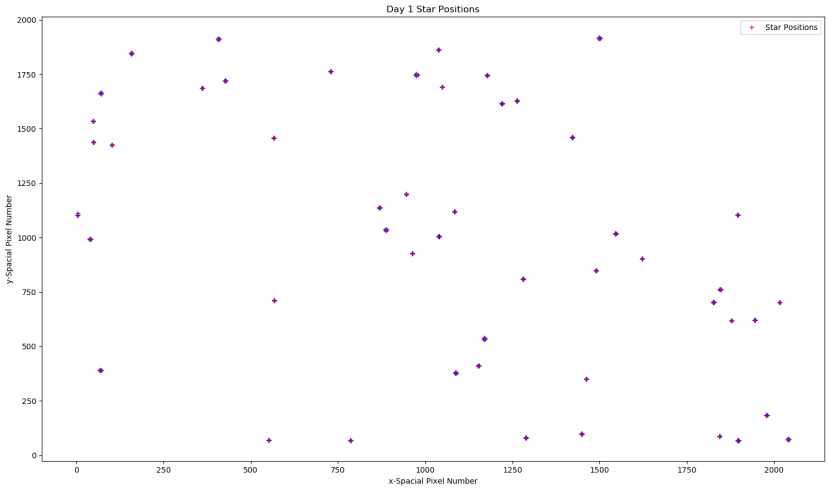

Figure 2: Star Positions For Day 1 Observations

4 Data Analysis and Modeling

4.1 Data Modeling

Recall that one of the main objectives for this lab was to cross-correlate star positions determined through Eq. 1-3 to the USNO star catalogue. This is because this procedure allows one to very easily distinguish 26-Prosperina from background stars. To do this, it was crucial to determine the star positions of all observation days prior to investigating anything about the correlation of stars with the USNO star catalogue. This calculation is thoroughly described in Section 3.

Now that the star positions of each observation day are present, it is feasible to conduct a respectable cross-correlation of stars with the USNO star catalogue. From Section 2, it was explained how the USNO star catalogue was interrogated and the Python programming code for this can be found under Appendix C. Cross-correlations were done for all four observation days. To cross-correlate the USNO stars with the calculated star positions, the standard coordinate transformation must be applied. This transformation is able to covert the coordinates of the USNO stars (given in RA [Deg] and DEC [Deg]) into x and y pixel positions. Once this is done, the results can be over plotted onto the plot displayed in Figure 2 and matches can be easily determined. The Python programming code for the standard coordinate transformation can be found under Appendix D and is described by the following set of equations.

This transformation requires two steps. First, RA and DEC coordinates must be projected onto a plane tangent to the section of the sky in question. This is done by the following equation where the ordered pair (X, Y) represent the projected coordinates, δ and α represent the declination and right ascension of USNO stars respectfully, and δ_0 and α_0 represent the parameters of centering for the USNO catalogue interrogation:

$$X = -\frac{\cos\delta \sin(\alpha - \alpha_0)}{\cos\delta_0 \cos\delta \cos(\alpha - \alpha_0) + \sin\delta \sin\delta_0} \qquad Y = -\frac{\sin\delta_0 \cos\delta \cos(\alpha - \alpha_0) - \cos\delta_0 \sin\delta}{\cos\delta_0 \cos\delta \cos(\alpha - \alpha_0) + \sin\delta \sin\delta_0} \tag{4}$$

Once the projected coordinates (X, Y) are determined, a second conversion to pixel positions (x, y) can be found through a straightforward algorithm. With the assumption of an ideal camera, the pixel conversion is given by the following equation where the ordered pair (x, y) represents the pixel position, the ordered pair (X, Y) represent the projected coordinates, f represents the focal length of the DIT, p represents the pixel size, the ordered pair (x_0, y_0) represent the centering pixel coordinates, and θ represents an allignment angle:

$$x = \frac{f}{p}(X\cos\theta - Y\sin\theta) + x_0 \qquad y = \frac{f}{p}(X\sin\theta + Y\cos\theta) + y_0 \tag{5}$$

Once the coordinates (x, y) for all USNO stars in question have been determined, it is possible to over plot these coordinates onto the plot which was displayed in Figure 2. The results are shown in Figure 3 and are discussed in Section 5.0.1.

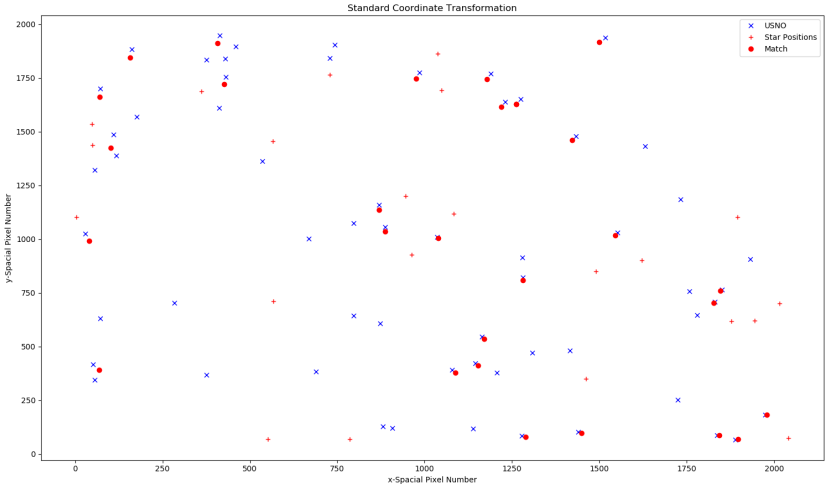

Figure 3: Star Positions Over Plotted With USNO Standard Coordinate Transformation For Day 1 Observations

4.2 Error Propagation

Note that the calculations performed for the standard coordinate transformation in Section 4.1 produce uncertainties. These uncertainties are explored through the method of least squares. To employ the method of least squares, the plate constants need to be determined. The plate constants refer to entries of a

transformation matrix which serves the purpose of transforming one set of coordinates to another (e.g, (X, Y) to (x, y)). Once the plate constants are found, discrepancies between the experimental transformation (Eq. 4-5) and predicted transformation (Eq. 8) can be compared and accounted for. The Python code for determining plate constants and conducting the following transformation can be found under Appendix E. The plate constants can be found by the following equations where a_{ij} represent plate constants, x_0 and y_0 represent offsets in pixels, $x_1, ..., x_n$ and $y_1, ..., y_n$ represent pixel positions, $X_1, ..., X_n$ and $Y_1, ..., Y_n$ represent projected coordinates, f represents the focal length of the DIT, and p represents the pixel size:

$$\mathbf{c_x} = (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{a_x} \qquad \text{where} \qquad \mathbf{a_x} = \begin{bmatrix} x_1 \\ . \\ . \\ x_n \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} (f/p)X_1 & (f/p)Y_1 & 1 \\ . & . & . \\ . & . & . \\ (f/p)X_n & (f/p)Y_n & 1 \end{bmatrix}, \qquad \mathbf{c}_x = \begin{bmatrix} a_{11} & a_{12} & x_0 \end{bmatrix} \tag{6}$$

$$\mathbf{c_y} = (\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T\mathbf{a_y} \qquad \text{where} \qquad \mathbf{a_y} = \begin{bmatrix} y_1 \\ . \\ . \\ y_n \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} (f/p)X_1 & (f/p)Y_1 & 1 \\ . & . & . \\ . & . & . \\ (f/p)X_n & (f/p)Y_n & 1 \end{bmatrix}, \qquad \mathbf{c_y} = \begin{bmatrix} a_{21} & a_{22} & y_0 \end{bmatrix} \tag{7}$$

When the plate constants are determined, it is possible to define a transformation matrix which maps coordinates of the form (X, Y) to coordinates of the form (x, y) and vice versa through the following equations:

$$\mathbf{x} = \mathbf{TX} \qquad \text{where} \qquad \mathbf{x} = (x, y, 1), \qquad \mathbf{X} = (X, Y, 1), \qquad \mathbf{T} = \begin{bmatrix} (f/p)a_{11} & (f/p)a_{12} & x_0 \\ (f/p)a_{21} & (f/p)a_{22} & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{8}$$

$$\mathbf{X} = \mathbf{T}^{-1}\mathbf{x} \qquad \text{where} \qquad \mathbf{x} = (x, y, 1), \qquad \mathbf{X} = (X, Y, 1), \qquad \mathbf{T} = \begin{bmatrix} (f/p)a_{11} & (f/p)a_{12} & x_0 \\ (f/p)a_{21} & (f/p)a_{22} & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{9}$$

The discussion of the results obtained by comparing Eq. 4-5 and Eq. 6-9 are more explored in Section 5.1.

4.3 26-Prosperina Tracking

By having cross-correlated stars from all four observation days, it is easy to determine the position of 26-Prosperina relative to background stars. Once 26-Prosperina has been located on a pixel grid, it is possible to get the respective RA and DEC coordinates by employing Eq. 9 or working backwards through Eq. 5-4. Once the respective RA and DEC coordinates have been determined for all four observation days, they can be plotted and the path of 26-Prosperina can be traced out through the sky. The Python code to determine this can be found under Appendix F. Knowing the duration of the conducted observations, the magnitude of proper motion can be resolved through the following equation where μ represents the magnitude of proper motion, μ_α and μ_δ represent the proper motion in the right ascension and declination directions respectfully, $\alpha_2 - \alpha_1$ and $\delta_2 - \delta_1$ represent the change in right ascension and declination respectfully, and Δt represents the duration of the conducted observations:

$$\mu = \sqrt{\mu_\delta^2 + \mu_\alpha^2 \cos^2(\delta_2 - \delta_1)} \qquad \text{where} \qquad \mu_\alpha = \frac{\alpha_2 - \alpha_1}{\Delta t}, \qquad \mu_\delta = \frac{\delta_2 - \delta_1}{\Delta t} \tag{10}$$

5 Discussion

5.0.1 Star Positions and Matching

The first step in determining the position and proper motion of 26-Prosperina was to find star positions and relevant matches for stars in the USNO star catalog. This was done for all four observation days provided by the DIT. Figure 3 depicts a star matching for the first observation day. Displayed in blue are the transformed USNO stars which were determined from Eq. 4-5. Shown as a red cross are the star positions determined through the center of light method as highlighted in Eq. 3, these are the exact stars as shown in Figure 2. The distinct red dot represents a match between a USNO star and a star position.

The reason that this process must be done is to determine which of the star positions in Figure 2 contains 26-Prosperina. From just determining star positions, it is ambiguous as to which set of coordinates belongs to a star and which set of coordinates belongs to an object other than a star. This is because without the proper knowledge of which objects are being observed, all objects appear to be points of light. However, if one were to have done a set of observation dealing with mapping distant stars, it would become very obvious as to which objects are distinct from these stars. This is due to the fact that distant stars appear to trace no movement through the sky, even during extended periods of observation. Now, for example, if an asteroid were to pass through a field of catalogued "fixed stars", it would trace a very evident path.

Luckily, much of the cataloguing of "fixed stars" has been done by the USNO. All that had to be done was to extract this catalogue from a patch of the sky which coincided to the observations carried out by the DIT. This information was accessible from any one of the files which were discussed in Section 2 through the extraction of a header. A header is a piece of information which describes the details of the observation process and is attached to a relevant `FITS` file. Through inspection from a corresponding header, the specified parameters could be determined and the USNO catalogue was able to be successfully interrogated and matched with star positions with an associated distance criterion error. This error is described in Section 5.0.2.

For further investigation, a method for determining better matches could have been employed. For example, although Eq. 5 is able to better align the USNO star data through an alignment angle, from Figure 3, it is apparent that some matches are not completely free of error. This alignment was done to ensure the most of amount of stars could receive matches with the calculated star positions. The following error is considered and acknowledged through the distance criterion.

5.0.2 Pixel Offset

The pixel offset is a simple error calculation to account for any misalignments from employing Eq. 5. This calculation is done independently for both the x and y pixels in order to clearly examine the pixel offset in both dimensions. The pixel error is determined through a distance criterion algorithm which simply determines the x and y distance from a USNO star to a star position after they have been over plotted. The results for some matched stars are shown in Table 2. It is important to acknowledge this error and to account for it. This is because when searching for 26-Prosperina, it is essential to know and store details of the exact positions of background stars in order to very clearly distinguish between a star versus an asteroid. If stars are misaligned, it could provide confusion as to which object is a star and which object is an asteroid since these objects will not have relevant matches to the USNO database.

Table 2: Pixel Offsets For Day 1 Observations.

Beginning of Data			
Match	USNO Correspondent	Pixel Offset In x	Pixel Offset In y
(1545.91,1017.88)	(1551.72,1029.42)	5.81	11.54
(1038.86,1006.10)	(1036.57,1008.81)	2.29	2.71
...	...	...	...
(1978.77,183.40)	(1974.45,181.46)	4.32	1.94
End of Data			

5.1 Residuals Between Measurement and Fit

An intermediate step in the process of determining the position and proper motion of 26-Prosperina was to settle and account for any transformation uncertainties. These uncertainties are not the same as the error described in Section 5.0.2. Here, what is discussed are the disparities between the experimental and predicted results for the conversion between standard coordinates to pixel coordinates and vice versa.

Recall, the experimental method for converting standard coordinates to pixel coordinates was to apply Eq. 4-5. The predicted results for such a transformation are outlined through the method of least squares described by Eq. 6-9. The reason that this process must be done is to examine the error in pixel for the experimental transformation. In order to do this, necessary plate constants had to be calculated. These plate constants are unique to the DIT, or in general, to the instrument of observation being employed. Plate constants are determined by measuring the standard coordinates for a minimum set of stars whose right ascensions and declinations are known and for which the standard coordinates can therefore be calculated. Here, we are able to determine the plate constants due to the fact that the cataloged stars from the USNO have known right ascensions and declinations. By making use of Eq. 6-7, the plate constants were determined and are displayed in Table 3.

Table 3: Calculated Plate Constants.

Beginning of Data	
Plate Constant	Calculated Value
a_{11}	0.999391
a_{12}	0.0348995
x_0	1024
a_{21}	-0.348995
a_{22}	0.999391
y_0	1024
End of Data	

The plate constants are important in order to construct the **T** matrix as shown in Eq. 8-9. This matrix is known as the affline transformation matrix and it's purpose is to transform elements from one set of coordinates to another. In general, an affline transformation between two vector spaces is defined through a linear transformation [1].

The least squares solution allows one to compute the pixel positions (x, y) from celestial coordinates (α, δ) (Eq. 8). The inverse of this transformation (Eq. 9) allows one to compute celestial coordinates (α, δ) from the pixel positions (x, y). This transformation is very useful to convert between the two coordinate systems and can be used as a basis for determining residuals. The results of applying the least square method provided the following pixel error as shown in Figure 4.

Interpreting the results of the least squares method, it can be seen that the error in pixel for both x and y range between -4 and 6. This entails is that the average uncertainty in experimental transformation (Eq. 4-5) is about 5 pixels. Here the negative sign in in front of the error in pixel simply means the predicted pixel is less than the experimental pixel. Being aware of this error is very important because this implies that the experimental transformation is not perfect. Of course with any experimental results, there will always be some experimental uncertainty which must be accounted for.

In this case, it is very useful to know exactly what the error in the pixel for the experimental mapping is. This is because if a USNO star and a star position do not match completely, instead of assuming the match is incorrect, we are able to study if the match is of the order of the error of that particular pixel in both the x and y directions. If this happens to be the case, we can consider the following a match since it lies in the bound of the experimental error. This is also the reason why some matches have been produced in Figure 3. These matches may not be completely precise, but they lie very well in the range of the experimental error.

Since the position and the proper motion of 26-Prosperina are also found using the same transformations, the uncertainties in these quantities can also be evaluated in terms of errors. This is extremely useful because it allows one to easily compute the uncertainties in the position and proper motion, which will be described as the transformed uncertainty in pixels into standard coordinates. This is discussed further in Section 5.2.

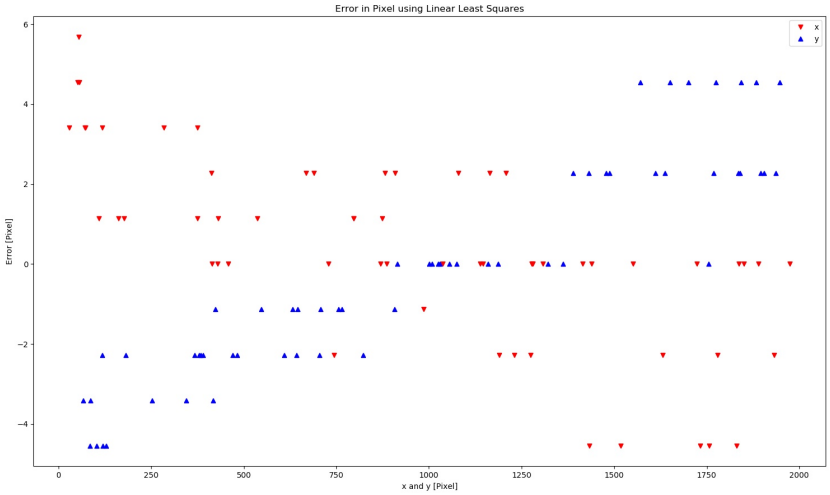

Figure 4: Error In Pixel Using The Method Of Least Squares

5.2 26-Prosperina Position and Proper Motion

The ultimate goal for this experiment was to determine the position and proper motion of 26-Prosperina. With all of the necessary steps completed, all that was left was to simply plot the transformed path of 26-Prosperina from it's pixel position to it's actual RA and DEC coordinates. The final results, estimated uncertainties, and proper motion calculation are provided in Figures 5-6 and Table 4 respectively.

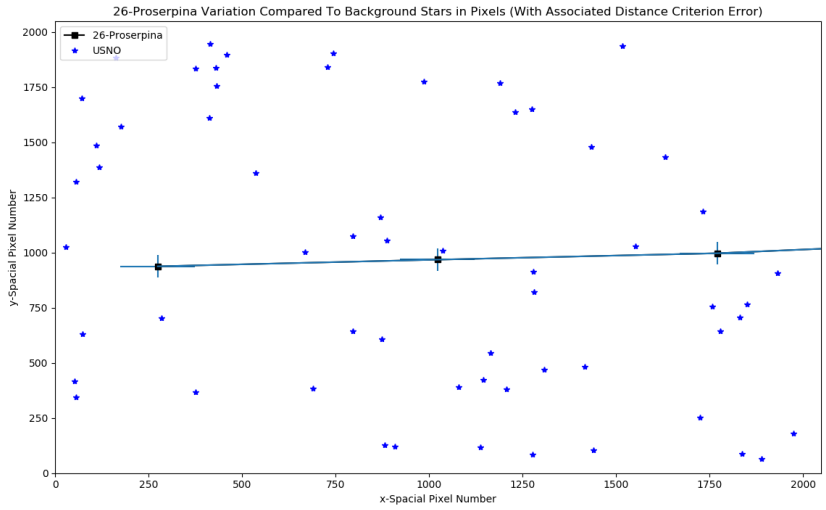

Figure 5: Pixel Position Of 26-Prosperina Compared To Background Stars

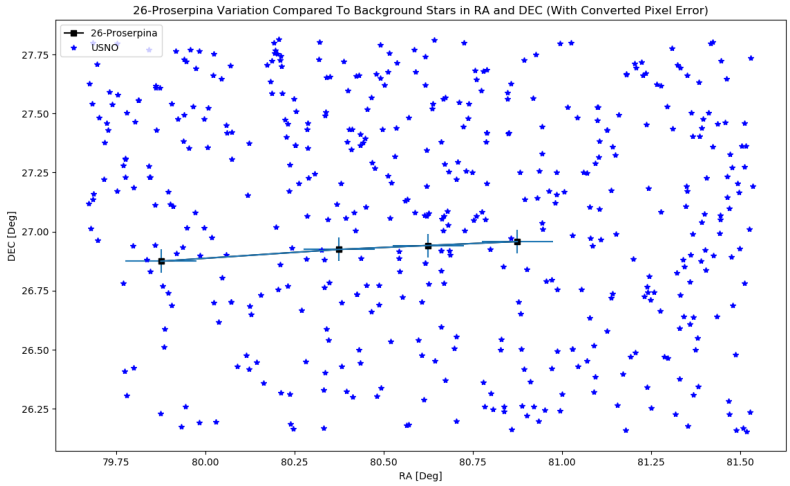

Figure 6: RA and DEC Position Of 26-Prosperina Compared To Background Stars

Table 4: Postion and Proper Motion Errors

Beginning of Data			
Error	Calculated Value	Proper Motion Error	Calculated Value
x Pixel	20.48	RA	2.5 [arcsec/hr]
y Pixel	15.36	DEC	1.25 [arcsec/hr]
RA [Deg]	0.10		
DEC [Deg]	0.07		
End of Data			

From Figure 5-6, the position and path of 26-Prosperina compared to background stars is very evident. The associated errors to each calculation is displayed in Table 4. By utilizing Eq. 10, the proper motion of 26-Prosperina was found to be 18.6 [arcsec/hr] with the following uncertainties for each component of the proper motion (RA and DEC) displayed in Table 4.

One notable observation of 26-Prosperina is that the magnitude of proper motion seems to decrease with time. The proper motion value provided is the average proper motion of all four observation days. For further research, this observation could be studied in more depth as it may allude to further properties of 26-Prosperina. Interpreting the final result is reasonable for the given uncertainties. This is because the uncertainties were determined through the method of least squares, as would be applied to any general pixel positions. The path of 26-Prosperina approximates a straight line in the sky which agrees with the fact that the errors in the x and RA directions seem to be higher than those in the y and DEC directions.

6 Conclusion

In conclusion, astrometric study of the position and proper motion of the asteroid 26-Prosperina was determined. Data for this experiment was provided through various institutions such as the DIT and the USNO. The given data for various days of observation was then reduced and star positions were cross correlated with the USNO star catalogue to determine the position of 26-Prosperina for each observation day. Multiple error prorogation methods such as a distance criterion and least squares method were employed and explained in order to effectively acknowledge any errors present while conducting this process. Overall, the magnitude of proper motion for 26-Prosperina was calculated to be 18.6 [arcsec/hr] with $\pm$ 2.5 [arcsec/hr] and $\pm$ 1.3 [arcsec/hr] uncertainties in the RA and DEC directions respectfully. The resulting position of 26-Prosperina is accurately displayed various plots with associated uncertainties.

7 Appendix

7.1 Appendix A Data Importation, Median, and Reduction Code

Below is a sample of code which was used to import and process various data sets of a file format known as FITS. Importation is shown for only one of each type file, but in general there where multiple files. This was done to avoid repetition. Also shown is the sample of code for determining medians of various data sets.

```
import astropy as ast
from astropy.io import fits
import numpy as np



#Data Importing


#Raw Data Import
DataA1=ast.io.fits.getdata(r'C:\Users\param\Desktop\Astro Labs\Lab 3\Data\
                           26Proserpina\20120119\26Prosperina-S001-R001-C001-r.fts')

#Flat Data Import
DataF1=ast.io.fits.getdata(r'C:\Users\param\Desktop\Astro Labs\Lab 3\Data\
                           26Proserpina\20120119\AutoFlat\AutoFlat-Dusk-r-Bin2-001.fts')

#Dark Data Import
DataB1=ast.io.fits.getdata(r'C:\Users\param\Desktop\Astro Labs\Lab 3\Data\
                           26Proserpina\20120119\Calibration\Bias-S001-R001-C001-B2.fts')



#Medians

#Flats Median
MedianFlats=np.zeros([2048,2048])
List=[]
for i in range (0, len(DataF1)):              #Loop which orders each element
    for j in range (0, len(DataF1[i])):       #from the list of flats into a list
        List.append(DataF1[i,j])              #and determines the median
        List.append(DataF2[i,j])              #and stores it
        List.append(DataF3[i,j])
        List.append(DataF4[i,j])
        List.append(DataF5[i,j])
        MedianFlats[i,j]=np.median(List)      #Create a new array to store
        List=[]                               #median values
np.savetxt('Day1MedianFlat.txt', MedianFlats) #Save as a txt file to access later

#Dark Median
MedianDarks=np.zeros([2048,2048])
List=[]
i=0
j=0
for i in range (0, len(DataD11)):
    for j in range (0, len(DataD11[i])):      #Loop which orders each element
        List.append(DataD1[i,j])              #from the list of darks into a list
```

```
        List.append(DataD2[i,j])              #and determines the median and
        List.append(DataD3[i,j])              #stores it
        List.append(DataD4[i,j])
        List.append(DataD5[i,j])
        List.append(DataD6[i,j])
        List.append(DataD7[i,j])
        List.append(DataD8[i,j])
        List.append(DataD9[i,j])
        List.append(DataD10[i,j])
        List.append(DataD11[i,j])
        List.append(DataD12[i,j])
        List.append(DataD13[i,j])
        List.append(DataD14[i,j])
        List.append(DataD15[i,j])
        MedianDarks[i,j]=np.median(List)    #Create a new array to store median
        List=[]                             #values
np.savetxt('Day1MedianDark.txt', MedianDarks) #Save as txt file to access later

#Corrected Median
Correct1=(DataA1-MedianDarks)/(MedianFlats-MedianDarks)
Correct2=(DataA2-MedianDarks)/(MedianFlats-MedianDarks)
Correct3=(DataA3-MedianDarks)/(MedianFlats-MedianDarks) #Apply the correction
MedianCorrect=np.zeros([2048,2048])                     #formula to all three
List=[]                                                 #raw sky data
i=0
j=0
for i in range (0, len(Correct1)):             #Loop which orders each element
    for j in range (0, len(Correct1[i])):      #from the list of corrected images
        List.append(Correct1[i,j])             #and determines the median and
        List.append(Correct2[i,j])             #stores it
        List.append(Correct3[i,j])
        MedianCorrect[i,j]=np.median(List)
        List=[]
np.savetxt('Day1Correct.txt', MedianCorrect) ##Save as txt file to access later
```

7.2 Appendix B Center Of Light Calculation Code

Below is a sample of code which was used to determine the star positions and centeroids of stars in corrected data sets.

```
#Star Positions



#Finding Stars
thresh=0.4
thresh2=0.2 #Create two threshhold to define minimim intensity to be a star
area=3 #Define an area around a star
xPos=[]
yPos=[]
xBoundStarL=[]
xBoundStarR=[]
yBoundStarD=[]
yBoundStarU=[] #Create multiple lists to store calculated information
for i in range(0, len(MedianCorrect)):            #Loop which determines if the
```

```
    for j in range (0, len(MedianCorrect[i])):    #position in the array is a
        if MedianCorrect[i,j]>thresh:             #star and if it is then store
            TF=False                              #its position and determine
            for p in range(i-area, i+area):       #fall out intensity area
                for q in range(j-area, j+area):
                    if MedianCorrect[i,j]<MedianCorrect[p,q]:
                        TF=True
            if TF==False:
                xPos.append(i)
                yPos.append(j)
                k=0
                while MedianCorrect[i, j-k]>thresh2:
                    k=k+1
                yBoundStarD.append(j-k)
                k=0
                while MedianCorrect[i, j+k]>thresh2:
                    k=k+1
                yBoundStarU.append(j+k)
                k=0
                while MedianCorrect[i-k, j]>thresh2:
                    k=k+1
                xBoundStarL.append(i-k)
                k=0
                while MedianCorrect[i+k, j]>thresh2:
                    k=k+1
                xBoundStarR.append(i+k)



#Centeroids

#x Centeroid
xCent=[]
xN=[]
xD=[] #Create multiple lists to store x centeroid position and other quantities
i=0
j=0
for i in range(0, len(xPos)):                          #Apply the center of light
    for j in range(xBoundStarL[i], xBoundStarR[i]):    #formula in the x direction
        xN.append(j*MedianCorrect[j, yPos[i]])         #and determine the x
        xD.append(MedianCorrect[j, yPos[i]])           #location of the centeroid
    xCent.append(sum(xN)/sum(xD))                      #and store it
    xN=[]
    xD=[]

#y Centroid
yCent=[]
yN=[]
yD=[] #Create multiple lists to store y centeroid position and other quantities
i=0
j=0
for i in range(0, len(yPos)):                          #Apply the center of light
    for j in range(yBoundStarD[i], yBoundStarU[i]):    #formula in the y direction
        yN.append(j*MedianCorrect[xPos[i], j])         #and determine the y
```

```
        yD.append(MedianCorrect[xPos[i], j])          #location of the centeroid
    yCent.append(sum(yN)/sum(yD))                     #and store it
    yN=[]
    yD=[]
```

7.3 Appendix C USNO Interrogation Function

Below is a sample of code which was used to call and interrogate data of star known star positions as catalogued by the USNO.

```
import matplotlib.pyplot as plt
import numpy as np



#USNO Interrogation Function



#Function
def usno(radeg,decdeg,fovam): # RA/Dec in decimal degrees/J2000.0 FOV in arc min.

    import urllib.request as url

    #url format for USNO
    str1 ='http://webviz.u-strasbg.fr/viz-bin/asu-tsv/?-source=USNO-B1'
    str2 ='&-c.ra={:4.6f}&-c.dec={:4.6f}&-c.bm={:4.7f}/{:4.7f}&-out.max=unlimited'.format(radeg,decdeg,
    #final URL: make sure it does not have any spaces or carriage returns/line feeds when copy-pasting
    sr = str1+str2

    # Read from the webpage, storing the page's contents in 's'.
    f = url.urlopen(sr)
    s = f.read()
    f.close()

    #column interpretation compass
    namecol, RAcol, DECcol, rband = 0, 1, 2, 12
    null1, null2 = '      ', ''

    #split webpage string into lines
    sl = s.splitlines()
    sl = sl[45:-1] # get rid of header
    name = np.array([])
    rad = np.array([]) # RA in degrees
    ded = np.array([]) # DEC in degrees
    rmag = np.array([]) # rmage
    #get data from each line
    for k in sl:
        kw = k.decode().split('\t')
        if kw[0] != '':
            name = np.append(name,kw[namecol])
            rad = np.append(rad,float(kw[RAcol]))
            ded = np.append(ded,float(kw[DECcol]))
            # deal with case where no mag is reported
            if (kw[rband] != null1) and (kw[rband] != null2):
```

```
                rmag = np.append(rmag,float(kw[rband]))
            else:
                rmag = np.append(rmag,np.nan)
    #return data
    return name,rad,ded,rmag



#Main
if __name__ == '__main__': #Call the USNO function
    name,rad,ded,rmag = usno(81.0333,26.9808,36.4) #Specify parameters
    w=np.where(rmag<13) #Take only the bright stars
    radw=rad[w]
    dedw=ded[w]
    plt.subplot(223)
    plt.title("USNO-B1 Stars")
    plt.plot(radw, dedw, 'bo', marker='x')
    plt.xlabel("RA [deg]")
    plt.ylabel("DEC [deg]")
    plt.tight_layout()
    plt.show() #Plot the USNO stars
```

7.4 Appendix D Standard Coordinate Transformation

Below is a sample of code which was used to experimentally apply the standard coordinate transformation to USNO stars and determine matches to found star positions.

```
import numpy as np



#Standard Coordinate Transformation



#Define sin and cos functions to convert degrees into radians
def sin(v):
    return np.sin(np.radians(v))

def cos(y):
    return np.cos(np.radians(y))



#X and Y
alpha0=81.0333
delta0=26.9808 #Define the following constants
X=[]
Y=[] #Create empty lists to store X and Y coordinates
i=0
for i in range(0, len(dedw)): #Loop which uses standard coordiante equation
    z1=cos(dedw[i])           #and stores this insides the lists previously
    z2=sin(radw[i]-alpha0)    #defined
    z3=cos(delta0)
    z4=cos(radw[i]-alpha0)
```

```
    z5=sin(dedw[i])
    z6=sin(delta0)

    X.append(((-1)*z1*z2)/((z3*z1*z4)+(z5*z6)))
    Y.append((1)*((z6*z1*z4)-(z3*z5))/((z3*z1*z4)+(z5*z6)))

    z1=0
    z2=0
    z3=0
    z4=0
    z5=0
    z6=0

#x and y
f=3454
p=0.018
x0=1024
y0=1024 #Define constants
x1=[]
y1=[] #Create empty lists to store x and y pixel positions
i=0
for i in range(0,len(X)):                                    #Loop which uses
    x1.append(((f/p)*((X[i]*cos(-2))-(Y[i]*sin(-2))))+x0)  #conversion formula
    y1.append(((f/p)*((X[i]*sin(-2))+(Y[i]*cos(-2))))+y0)  #and stores this inside
i=0                                                          #lists previously
j=0                                                          #defined
Matchx=[]
Matchy=[] #Create empty lists to store matches between USNO and star positions
for i in range (0, len(x1)): #Loop which determines matches
    for j in range (0, len(yCent)):
        if abs(x1[i]-yCent[j])<40 and abs(y1[i]-xCent[j])<40:
            Matchx.append(yCent[j])
            Matchy.append(xCent[j])
```

7.5 Appendix E Plate Constants and Predicted Transformation

Below is a sample of code used to determine pixel error. Here, the plates constants are found and the predicted transformation is applied to determine pixel error.

```
import numpy as np



#Predicted Transformation and Pixel Error



#Plate Constants

#x
xCol=np.zeros([len(x1),1])
i=0
for i in range (0,len(x1)): #Convert x pixel positions list into a vector
    xCol[i,0]=x1[i]
bMat=np.zeros([len(X),3]) #Define the matrix which is used to determine plate
```

```
i=0                         #constants
for i in range (0, len(bMat)):
    for j in range (0, 1):
        bMat[i,j]=((f/p)*X[i]) #Fill in corresponding entries into this matrix
i=0
for i in range (0, len(bMat)):
    for j in range (1, 2):
        bMat[i,j]=((f/p)*Y[i]) #Fill in corresponding entries into this matrix
i=0
for i in range (0, len(bMat)):
    for j in range (2, 3):
        bMat[i,j]=1 #Fill in corresponding entries into this matrix
Firstx=np.linalg.inv(np.dot(bMat.transpose(), bMat))
Secondx=np.dot(bMat.transpose(), xCol)
Constantsx=np.dot(Firstx, Secondx) #Preform linear algebra calculations to determine
                                   #plate constants from x
#y
yCol=np.zeros([len(y1),1])
i=0
for i in range (0, len(y1)): #Convert y pixel positions list into a vector
    yCol[i,0]=y1[i]
Firsty=Firstx
Secondy=np.dot(bMat.transpose(), yCol) #Perform the same linear algebra calculations
Constantsy=np.dot(Firsty, Secondy)     #to determine plate constants from y



#Error In Pixel
a11=Constantsx[0,0]
a12=Constantsx[1,0]
x0P=Constantsx[2,0]
a21=Constantsy[0,0]
a22=Constantsy[1,0]
y0P=Constantsy[2,0] #Define plate constants as decribed in the lab handout
ratio=(f/p) #Define this ratio beforehand to shorten code
T=np.zeros([3,3]) #Define T matrix for transformation
T[0,0]=(ratio*a11)
T[1,0]=(ratio*a21)
T[2,0]=0
T[0,1]=(ratio*a12)
T[1,1]=(ratio*a22)
T[2,1]=0
T[0,2]=x0P
T[1,2]=y0P
T[2,2]=1 #Fill in T matrix with corresponding entries
x1P=[]
y1P=[]
garbage1s=[] #Create lists to store predicted pixel positions
XVec=np.zeros([3,1])
xTemp=np.zeros([3,1])
i=0
for i in range(0,len(x1)): #Loop which determines predicted pixel positions and
    XVec[0,0]=X[i]         #stores them in previously defined lists
    XVec[1,0]=Y[i]
```

```
    XVec[2,0]=1
    xTemp=np.dot(T,XVec)
    x1P.append(xTemp[0,0])
    y1P.append(xTemp[1,0])
    garbage1s.append(xTemp[2,0])
    XVec=np.zeros([3,1])
    xTemp=np.zeros([3,1])
Errorx=[]
Errory=[] #Create lists to store the error in pixel
i=0
for i in range (0,len(x1P)):      #Loop which determines the error in pixel
    Errorx.append((x1[i]-x1P[i])) #for each pixel direction
    Errory.append((y1[i]-y1P[i]))
```

7.6 Appendix F 26-Proserpina Position and Proper Motion

Below is a sample of code which was used to determine the position and proper motion of 26-Proserpina.

```
import numpy as np



#Asteroid Position and Proper Motion



#Define sin and cos functions to convert degrees into radians
def sin(v):
    return np.sin(np.radians(v))

def cos(y):
    return np.cos(np.radians(y))



#Position

#X and Y
AsteroidPositionRA=[80.8750,80.6250,80.3750,79.8750]
AsteroidPositionDEC=[26.9583,26.9417,26.9250,26.8750] #Use data from all 4 days
alpha0=80.6250                                        #to store asteroid position
delta0=26.9250
AsteroidX=[]
AsteroidY=[] #Create empty lists to store the standard coordinates
i=0
for i in range(0, len(AsteroidPositionDEC)): #Loop to determine the standard coordinates
    q1=cos(AsteroidPositionDEC[i])           #of the asteroid
    q2=sin(AsteroidPositionRA[i]-alpha0)
    q3=cos(delta0)
    q4=cos(AsteroidPositionRA[i]-alpha0)
    q5=sin(AsteroidPositionDEC[i])
    q6=sin(delta0)

    AsteroidX.append(((-1)*q1*q2)/((q3*q1*q4)+(q5*q6)))
    AsteroidY.append((1)*((q6*q1*q4)-(q3*q5))/((q3*q1*q4)+(q5*q6)))
```

```
    q1=0
    q2=0
    q3=0
    q4=0
    q5=0
    q6=0

#x and y
Ax1=[]
Ay1=[] #Create empty lists to store the pixel position
i=0
for i in range(0,len(AsteroidX)): #Loop to determine pixel position of the asteroid
    Ax1.append(((f/p)*((AsteroidX[i]*cos(-2))-(AsteroidY[i]*sin(-2))))+x0)
    Ay1.append(((f/p)*((AsteroidX[i]*sin(-2))+(AsteroidY[i]*cos(-2))))+y0)
xPixelErr=100
yPixelErr=50 #Account for the error
```

References

[1] Lab 3: Astrometry from CCD Images. http://www.astro.utoronto.ca/ astrolab/.

[2] AST325/326 Course Website. http://www.astro.utoronto.ca/ astrolab/.

[3] Dae-Sik Moon. Astrometry. http://www.astro.utoronto.ca/ astrolab/.