The SUSI Astrometric Model and "the pointing problem"

W. J. Tango

25 April 2005

Abstract

These notes detail the theory behind the siderostat pointing model used in SUSI and discuss "the pointing problem." Although the theory is available in other places it's not very accessible. The "pointing problem" refers to the fact that the accuracy of pointing the siderostats has never been as good as it should be. Reasons for this are discussed here.

The program used to determine the pointing solution has been completely rewritten. Instructions for using it in the companion document *How to use "susipoint" to find siderostat pointing solutions.*

1 Introduction

The SUSI Astrometric Model (AM) is the mathematical model of the instrument. It is implemented by the `astromod` daemon that runs on the main SUSI control computer, and one of its tasks is to determine the pointing of the siderostats.

There are four data files associated with the AM. They are text files and data can be extracted using standard SUSI library routines. These reside in `$SUSI/etc`, where `$SUSI` is a symbolic name for the SUSI root directory on the control computer. The files are:

starlist is a database of star coordinates. Stars are indexed by their HR (Bright Star Catalog) number. The "HR numbers" in the range 9900 to 9923 are available for test purposes. The HR number 99hh points to a fictitious star having the specified hour angle.

sidstats is the database of siderostat parameters. This file is regularly updated because it includes the autocollimation positions, which can change slightly over time.

baselns is similar to the siderostat database, but contains the x, y and z coordinates of each baseline as well as the associated white light fringe position. The file needs updating whenever there are changes to the optical path due to realignment of optics, etc.

system contains the latitude and longitude used by the AM. These should never be changed (any errors in latitude and longitude will appear as small errors in the baseline coordinates). The three parameters called POLARX, POLARY and DUT are respectively the x and y coordinates of the Earth's axis of rotation and the current error between UTC and "true" Universal Time. The parameters are published by the US Naval Observatory and should be updated on a weekly basis.

All changes to the files are made using a version control system. Historically we used `SCCS` but now use `CVS`. The autocollimation positions are normally updated at the beginning of each night and this can be done from the SUSI scheduler.[1]

There is a program, `susipoint`, that can be used to estimate the siderostat model parameters from observational data and thereby improve the pointing.

In practice we have found that the "pointing" of the siderostats is not particularly good. This has been an ongoing nuisance but as we extend the instrument to longer baselines it is becoming a more pressing issue. Why is the pointing so poor? Possible reasons include:

- Software timing errors. The "pointing solution" is calculated from observed data listing the actual and computed siderostat positions. The local siderostat controllers have no knowledge of the time, so there is no guarantee that the "actual" and "computed" positions refer to the same instant of time.
- Errors in the AM. As we have seen, the AM is implemented by a program called `astromod`. If this contains errors the pointing will be affected.
- Errors in the analysis software. The `susipoint` program is used to determine corrections to the current model. It may also contain errors.
- Significant sources of error may not have been included in the model. The most obvious of these is periodic gear error associated with the azimuth and elevation drives.
- Possible systematic error in the determination of the autocollimating positions.
- Incorrect shaft encoder readings. Appendix B describes how the shaft encoders are used. The encoders sometimes report erroneous values. These errors occur sporadically and the most likely cause is electrical noise or interference. Erroneous points can corrupt the data recorded for pointing solutions, leading to very large residuals.

  Mike has recently analyzed some pointing data and has shown that these sporadic errors are probably the most likely cause of poor pointing solutions. Fortunately the discrepant points are relatively easy to identify and can be removed from the data set.
- Hardware problems associated with the mechanical alignment and operation of the siderostats.

The original AM was developed by the author, Theo ten Brummelaar, Rob Minard and Erik Thorvaldson. The basic reference is a document by WJT ("Astrometric notes for the Sydney University Stellar Interferometer") written sometime in the late 1980s, and Erik's honours thesis (1991). None of these documents is easily accessible, and these notes summarize their contents.

The vector method used for the SUSI AM is based on the formalism developed by C.A. Murray C.A. in his monograph *Vectorial Astrometry*, Adam Hilger, London, 1983, although I have not adopted Murray's rather cumbersome notation.[2]

[1] Mike Ireland has written a script called `susiedit` that facilitates this process.

[2] In classical mechanics a coordinate transformation is mathematically indistinguishable from a rotation of a solid body, although the two operations are physically very different. A coordinate transformation does not result in a physical change to the system while a rotation clearly changes the relative positions of a body with

2 The `astromod` daemon

The `astromod` program is a daemon that detaches itself from the user's environment and runs in the background. Its job is to produce and update a file containing the siderostat and path compensator positions and rates. This file is updated in "batches:" `astromod` sleeps most of the time but every 5 minutes it wakes up, calculates another bunch of numbers, and goes back to sleep. This rather strange behavior reflects the limited computing power available when SUSI first came on line.

When a new batch is calculated a the apparent and observed positions of the target are calculated using `SLALIB`. This a library of astrometric functions written by Patrick Wallace and is essentially a canonical realization of the IAU reference system.

Each batch consists of pointing data calculated at 10 second intervals. "Client" applications that use the data may need to interpolate; ten seconds was chosen to ensure that linear interpolation would always be adequate. It should also be noted that many of the control systems in SUSI are *rate* servos. The rates change very slowly and no interpolation is required. The data most relevant to the siderostat pointing are the current star azimuth and elevation and the model positions (azimuth and elevation) of the two siderostats. This output file is called `astromod.dat` and is located in `$SUSI/tmp/`.

The daemon can be started with a wide range of options. With a few exceptions, all the information needed by the daemon can be supplied as a command line string. The most notable exception is the autocollimation mode.[3]

3 Siderostat pointing

Let $\hat{\mathbf{m}}$ be a unit vector corresponding to the normal to a given siderostat mirror. The author and R. A. Minard have shown that

$$\hat{\mathbf{m}} = \frac{\hat{\mathbf{s}} + \hat{\mathbf{a}}}{|\hat{\mathbf{s}} + \hat{\mathbf{a}}|} = \frac{\hat{\mathbf{s}} + \hat{\mathbf{a}}}{[2(1 + \hat{\mathbf{s}} \cdot \hat{\mathbf{a}})]^{1/2}} \tag{3.1}$$

where $\hat{\mathbf{a}}$ is the unit vector corresponding to the *autocollimating* direction for the siderostat and $\hat{\mathbf{s}}$ is the unit vector corresponding to the star's "observed direction" (i. e., the direction corrected for atmospheric refraction). Eq. (3.1) is the vector version of the law of reflection.

It should be emphasized that Eq. (3.1) is a vector equation and is independent of any particular coordinate frame. In practice, however, we always need to calculate the Cartesian components of the vectors with respect to some particular frame of reference. We assume that the star's position and the autocollimating position are known in the local zenith frame of reference (see Appendix A). Eq. (3.1) allows us to calculate the mirror position in this reference frame. The azimuth and elevation of the siderostats are *approximately* equal to the local zenith azimuth and elevation, but will differ because of instrumental factors.

The problem, then, is to find a procedure that will relate the actual siderostat shaft encoder readings to the true azimuth and elevation in the local zenith frame. The method

respect to its surroundings. Murray uses a notation, originally due to Milne, that distinguishes between these two operations. In the the present context we only use coordinate transformations, so there is no need to introduce a special notation.

[3] It would not be difficult to change this behavior; at the moment there is no pressing need but it would need to be done if autocollimation were to be included in `scheduler`.

I developed is to define a mirror frame of reference which has the mirror normal as one of its basis vectors. Then, by a series of coordinate transformations, the frame is rotated until it is coincident with the local zenith frame. Applying the same set of transformations to the mirror normal will then give its coordinates in the local zenith frame. This procedure is most easily explained by considering a perfect siderostat.

3.1 Ideal siderostats

An ideal or perfect siderostat has its azimuth axis pointing to the zenith, the elevation axis orthogonal to the azimuth axis, and the mirror normal perpendicular to the elevation axis. The shaft encoders are assumed to read the true angular position of the two axes.

Define a "mirror coordinate frame" having the unit vectors $(\hat{\mathbf{x}}\ \hat{\mathbf{y}}\ \hat{\mathbf{z}})$ where $\hat{\mathbf{y}}$ corresponds to the mirror normal. In this reference frame the mirror position will be

$$\hat{\mathbf{m}} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{3.2}$$

The unit vector $\hat{\mathbf{x}}$ is coincident with the siderostat elevation axis $\hat{\mathbf{e}}_s$ and $\hat{\mathbf{z}}$ is chosen to complete the right-handed basis for the coordinate frame. It will lie in the plane of the mirror and is perpendicular to both the mirror normal and the elevation axis.

We define a second frame by a rotation about the x-axis. The angle of rotation will be ϵ_s, where the subscript denotes that this is the siderostat elevation angle–i. e., the angle corresponding to the physical rotation of the elevation axis. In the new frame of reference $\hat{\mathbf{z}}$ will point to the zenith. The corresponding transformation matrix will be

$$\mathbf{T}'_e(\epsilon_s) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\epsilon_s & -\sin\epsilon_s \\ 0 & \sin\epsilon_s & \cos\epsilon_s \end{bmatrix} \tag{3.3}$$

(the dash notation will be explained below).

We next transform to a third coordinate frame by a rotation about the z-axis. The transformation matrix is

$$\mathbf{T}_a(a_s) = \begin{bmatrix} \cos a_s & \sin a_s & 0 \\ -\sin a_s & \cos a_s & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

where a_s is the siderostat azimuth.

The elevation transformation corresponds to a counterclockwise rotation of the reference frame while the azimuth transformation is clockwise. This is accordance with the standard definition of these angles (one reason for using the zenith distance $\zeta = \pi/2 - \epsilon$ is that the azimuth and zenith distance both increase in a clockwise sense). Counterclockwise transformations will be indicated by a dash symbol ($'$). A counterclockwise rotation obviously undoes a clockwise rotation about the same axis, so $\mathbf{T}'\mathbf{T} = \mathbf{1}$; i. e., $\mathbf{T}'$ is the inverse of $\mathbf{T}$.

By virtue of our assumption that the siderostat is ideal the last reference frame is identical to the local zenith frame and the mirror normal, expressed in the local zenith frame, will

Symbol	Description
$\hat{\mathbf{z}}_s$	The azimuth axis of the siderostat, which is assumed to point upwards.
$\hat{\mathbf{e}}_s$	The elevation axis. The direction of $\hat{\mathbf{e}}_s$ is chosen so that when the siderostat azimuth is zero $\hat{\mathbf{e}}_s$ points eastwards.
$\hat{\mathbf{m}}$	The siderostat mirror normal. It points outwards from the mirror.
$\hat{\mathbf{a}}_{N,S}$	The optical axis vectors. They point in the direction of the propagation of light from the siderostat mirror into the instrument. There are two such vectors, corresponding to the north-looking and south-looking configurations of the siderostat.

Table 3.1: The six unit vectors associated with each siderostat. There are two autocollimating positions, depending on whether the siderostat is looking north or south.

be

$$\hat{\mathbf{m}} = \begin{bmatrix} \cos a_s & \sin a_s & 0 \\ -\sin a_s & \cos a_s & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\epsilon_s & -\sin\epsilon_s \\ 0 & \sin\epsilon_s & \cos\epsilon_s \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} \cos\epsilon_s \sin a_s \\ \cos\epsilon_s \cos a_s \\ \sin\epsilon_s \end{bmatrix} \tag{3.5}$$

This result should be compared to the left-hand side of Eq. (A.3): the siderostat azimuth and elevation are in this ideal case equal to the azimuth and elevation in the local zenith frame.

3.2 Real siderostats

Real siderostats will not have their axes perfectly aligned, there will be errors associated with the encoders, etc.

There are five unit vectors (or directions) associated with each siderostat. These are summarized in Table (3.2). The autocollimating position $\hat{\mathbf{a}}$ in Eq. (3.1) will correspond to either $\hat{\mathbf{a}}_N$ or $\hat{\mathbf{a}}_S$ depending on the operating mode of the siderostat (north- or south-looking).

We can again define a mirror coordinate frame. The y-axis is chosen to be coincident with the mirror normal and the unit vector $\hat{\mathbf{z}}$ is defined to be

$$\hat{\mathbf{z}} = \frac{\hat{\mathbf{e}}_s \times \hat{\mathbf{y}}}{|\hat{\mathbf{e}}_s \times \hat{\mathbf{y}}|} \tag{3.6}$$

The unit vector $\hat{\mathbf{x}}$ is chosen to form a right-handed basis set. The x-axis will be approximately coincident with the elevation axis and $\hat{\mathbf{z}}$ lies in the plane of the mirror.

We now apply a series of transformations to take us from the mirror reference frame to the local zenith frame. It should be emphasized that these transformations do not represent physical rotations: the mirror is assumed to be fixed and pointing in some direction. Except for the azimuth and elevation rotations, which conform to standard astronomical convention, the sense of the various rotations is arbitrary. I have used counterclockwise rotations. We know the mirror coordinates in the mirror reference frame–they are given in Eq. (3.2). We then apply a series of coordinate transformations until we are in the local zenith frame as follows:

1. Let δ_2 be the angular misalignment between the x-axis in the mirror reference frame and $\hat{\mathbf{e}}_s$. The first intermediate frame is found by a counterclockwise rotation δ_2 about

the $\hat{\mathbf{z}}$ direction. This will bring the x-axis of the new frame into coincidence with the elevation axis. The transformation matrix will be

$$\mathbf{T}'_7(\delta_2) = \begin{bmatrix} \cos\delta_2 & -\sin\delta_2 & 0 \\ \sin\delta_2 & \cos\delta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

2. The next coordinate frame is obtained by rotating around the x-axis until the y-axis lies in the plane of $\hat{\mathbf{z}}_s$ and $\hat{\mathbf{e}}_s$. The angle will be denoted by ϵ_s and it is assumed that

$$\epsilon_s = \epsilon_e - \Delta\epsilon \tag{3.8}$$

where ϵ_e is the elevation shaft encoder reading and $\Delta\epsilon$ the encoder offset. It should be noted that Eq. (3.8) is the simplest possible model for the readout system. Appendix B discusses this point in more detail.

The transformation matrix is

$$\mathbf{T}'_6(\epsilon_s) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\epsilon_s & -\sin\epsilon_s \\ 0 & \sin\epsilon_s & \cos\epsilon_s \end{bmatrix} \tag{3.9}$$

3. The elevation and azimuth axes may not be perpendicular. The orthogonality error will be δ_1, defined by

$$\sin\delta_1 = -\hat{\mathbf{z}}_s \cdot \hat{\mathbf{e}}_s \tag{3.10}$$

The next coordinate frame is obtained by a rotation of δ_1 around the y-axis, bringing the z-axis into coincidence with $\hat{\mathbf{z}}_s$, the physical azimuth axis of the siderostat. The corresponding transformation is

$$\mathbf{T}'_5(\delta_1) = \begin{bmatrix} \cos\delta_1 & 0 & \sin\delta_1 \\ 0 & 1 & 0 \\ -\sin\delta_1 & 0 & \cos\delta_1 \end{bmatrix} \tag{3.11}$$

4. The next frame of reference is the "siderostat coordinate frame." The x-axis is defined by

$$\hat{\mathbf{x}}_s = \frac{\hat{\mathbf{y}} \times \hat{\mathbf{z}}_s}{|\hat{\mathbf{y}} \times \hat{\mathbf{z}}_s|} \tag{3.12}$$

where $\hat{\mathbf{y}}$ corresponds to the y-axis in the local zenith frame (it lies in the horizontal plane and points North). The unit vector $\hat{\mathbf{y}}_s$ is defined as usual to complete a right-handed basis set and is approximately coincident with $\hat{\mathbf{y}}$. The previous frame of reference brought the z-axis into alignment with $\hat{\mathbf{z}}_a$. The next transformation is a rotation through an angle a_s to bring the y-axis into coincidence with $\hat{\mathbf{y}}_s$:

$$\mathbf{T}_4(a_s) = \begin{bmatrix} \cos a_s & \sin a_s & 0 \\ -\sin a_s & \cos a_s & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.13}$$

As for the elevation, the azimuth angle a_s is modelled by

$$a_s = a_e - \Delta a \tag{3.14}$$

where a_e is the azimuth encoder reading and Δa the azimuth offset.

5. The cumulative transformations so far will give the mirror coordinates in the "siderostat coordinate frame." This frame is almost, but not exactly, coincident with the local zenith frame because the azimuth axis may be tilted with respect to the local vertical. Two angles are required to specify the tilt. We define θ_1 to be the angle between $\hat{\mathbf{z}}$ and $\hat{\mathbf{z}}_s$. The $(x_s\ y_s)$ plane will of course be tilted by the same amount with respect to the local horizon plane. The intersection of the two planes defines the line of nodes, and we define θ_2 to be the angle between the line of nodes and the x-axis. The transformation that will transform from the siderostat reference frame to the local zenith frame can be done in three steps (the procedure is based on the classical method of Euler angles):

   - A counterclockwise rotation about the z_s axis to bring the x-axis into coincidence with the line of nodes.
   - A rotation about the x-axis to bring the z_s-axis into coincidence with the local zenith direction $\hat{\mathbf{z}}$.
   - A clockwise rotation about the z-axis to restore the x-axis to its original orientation.

   The third transformation is simply the inverse of the first, and the three transformations can be written as

$$\mathbf{T}(\theta_1,\theta_2) = \mathbf{T}_1(\theta_2)\mathbf{T}_2(\theta_1)\mathbf{T}'_3(\theta_2) = \begin{bmatrix} \cos\theta_2 & \sin\theta_2 & 0 \\ -\sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & \sin 0 & 0 \\ 0 & \cos\theta_1 & \sin\theta_1 \\ 0 & -\sin\theta_1 & \cos\theta_1 \end{bmatrix} \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.15}$$

The final coordinate frame is the local zenith coordinate frame. The azimuth and elevation of the siderostat mirror in the local zenith frame will be:

$$\begin{bmatrix} \cos\epsilon\sin a \\ \cos\epsilon\cos a \\ \sin\epsilon \end{bmatrix} = \mathbf{T}_1(\theta_2)\mathbf{T}_2(\theta_1)\mathbf{T}'_3(\theta_2)\mathbf{T}_4(a_s)\mathbf{T}'_5(\delta_1)\mathbf{T}'_6(\epsilon_s)\mathbf{T}'_7(\delta_2) \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \tag{3.16}$$

For convenience, the angles used to describe the siderostat model are summarized in Table (3.2).

4 Calculation of the siderostat azimuth and elevation

Assuming that the angles in Table (3.2) are known, Eqs. (3.8, 3.14, 3.16) give a recipe that relates the siderostat encoder azimuth and elevation, a_e and ϵ_e, to the azimuth and elevation in the local zenith frame.

The computational task is to develop algorithms that will answer the following two problems:

- Given the encoder azimuth and elevation of the siderostats, what are the coordinates of $\hat{\mathbf{m}}$ in the local zenith frame?

Symbol	Description
δ_1	The orthogonality error between the azimuth and elevation axes. The angle between the axes is $90^\circ + \delta_1$.
δ_2	The angular misalignment between the siderostat mirror and the elevation axis. It is measured clockwise from the elevation axis to the plane of the mirror.
$\Delta\epsilon$	The elevation shaft encoder offset.
Δa	The azimuth shaft encoder offset.
θ_1	The tilt between the siderostat azimuth axis and the local vertical.
θ_2	The "line of nodes" specifying the orientation of the azimuth axis tilt.

Table 3.2: The angles used to specify the siderostat model.

- Given the coordinates of $\hat{\mathbf{m}}$ in the local zenith frame, what are the corresponding encoder readings?

The second question is obviously just the inverse of the first, but requires a somewhat more complicated algorithm.

Recall that the star position $\hat{\mathbf{s}}$ will be known in the local zenith frame. Before we can calculate the mirror position in the local zenith frame we must determine the autocollimation position in the local zenith frame. The autocollimation positions are determined in terms of shaft encoder readings as part of the nightly setup procedure for SUSI. Finding the autocollimation positions in the local zenith frame is an example of the first problem mentioned above and we find them by a direct application of Eqs. (3.8, 3.14, 3.16). In the case of Eq. (3.16) everything on the righthand side is known and it is evaluated using standard matrix algebra. In the AM code this is done by a function called `azel2xyz`. It takes as input the measured azimuth and elevation positions and returns the Cartesian coordinates of the vector in the local zenith frame.

Given the autocollimation vector in the local zenith frame, the mirror vector in the same frame is determined using Eq. (3.1). To find the encoder azimuth and elevation angles we need to solve the second problem mentioned above.

Eq. (3.16) can be written as

$$\mathbf{T}_3(\theta_2)\mathbf{T}'_2(\theta_1)\mathbf{T}'_1(\theta_2)\begin{bmatrix}\cos\epsilon\sin a\\ \cos\epsilon\cos a\\ \sin\epsilon\end{bmatrix} = \mathbf{T}_4(a_s)\mathbf{T}'_5(\delta_1)\mathbf{T}'_6(\epsilon_s)\mathbf{T}'_7(\delta_2)\begin{bmatrix}0\\1\\0\end{bmatrix} \tag{4.1}$$

The left hand side of this equation can be calculated numerically. We will call the resulting vector $\mathbf{v}$:

$$\mathbf{v} = \begin{bmatrix}v_x\\ v_y\\ v_z\end{bmatrix} = \mathbf{T}_3(\theta_2)\mathbf{T}'_2(\theta_1)\mathbf{T}'_1(\theta_2)\begin{bmatrix}\cos\epsilon\sin a\\ \cos\epsilon\cos a\\ \sin\epsilon\end{bmatrix} \tag{4.2}$$

The righthand side must be evaluated algebraically:

$$\mathbf{T}_4(a_s)\mathbf{T}'_5(\delta_1)\mathbf{T}'_6(\epsilon_s)\mathbf{T}'_7(\delta_2)\begin{bmatrix}0\\1\\0\end{bmatrix} = \begin{bmatrix}f\cos a_s + \sin a_s\cos\epsilon_s\cos\delta_2\\ -f\sin a_s + \cos a_s\cos\epsilon_s\cos\delta_2\\ \sin\delta_1\sin\delta_2 + \sin\epsilon_s\cos\delta_1\cos\delta_2\end{bmatrix} \tag{4.3}$$

where

$$f = \sin\epsilon_s \sin\delta_1 \cos\delta_2 - \cos\delta_1 \sin\delta_2 \tag{4.4}$$

The elevation can now be found from the last row of Eq. (4.3):

$$\epsilon_s = \arcsin\left[\frac{v_z - \sin\delta_1 \sin\delta_2}{\cos\delta_1 \cos\delta_2}\right] \tag{4.5}$$

The first two rows of Eq. (4.3) can be used to find the azimuth:

$$f\cos a_s + \sin a_s \cos\epsilon_s \cos\delta_2 = v_x \tag{4.6}$$

$$-f\sin a_s + \cos a_s \cos\epsilon_s \cos\delta_2 = v_y \tag{4.7}$$

This is a pair of linear equations in the unknowns $\sin a_s$ and $\cos a_s$ and the solution is straightforward; in `astromod` the "`atan2`" function is used to calculate a_s unambiguously.

This procedure for determining the siderostat encoder angles from the Cartesian components of a vector in the local zenith frame is implemented in the AM code by a function called `xyz2azel`.

I have re-examined the `astromod` code and as far as I can tell the code implements the procedures given in these notes.

5 Determination of the model parameters

The various parameters fall into four groups:

The encoder offsets For various practical reasons it is difficult to set the encoders very accurately. As a result the offset error can be of the order of $1^\circ \sim 2^\circ$.

The alignment errors δ_1 and δ_2 These errors are a consequence of the fabrication process. The manufacturing tolerances were very tight, and it is reasonable to assume that these two angles will be small.

The azimuth axis tilt The tilt (both magnitude and orientation) can be be measured directly. Erik used a theodolite to determine the tilt parameters, but the Talyvel (a precision electronic level) can also be employed and is probably easier to use than the theodolite.

The autocollimation angles These are determined experimentally as part of the nightly alignment procedure. We have implicitly assumed that this procedure is unbiased, but it is straightforward to include autocollimation azimuth and elevation offsets into the fitting process. This also has the advantage that data using north- and south-looking configurations can be combined for an "all sky" solution.

Pat Wallace sells a program called `tpoint` that is designed to find pointing solutions for telescopes. He has indicated that he would be prepared to modify it for our requirements (for a price!). However, a comment in the `susipoint` code states that the "least squares fitting in this program is based on the program 'TPOINT' written by P.T. Wallace. . . ."

The original `susipoint` was used to estimate the parameters from observational data. "Pointing data" are sets of model predictions and actual shaft encoder readings. As noted previously, because the local siderostat controllers have no knowledge of the time there can be timing errors so the model predictions and encoder readings may actually refer to different times. Although the controllers still don't have access to time information, the new serial port hardware greatly reduces this problem.

The pointing files used by `susipoint`, and which have been adapted for the current system, contain the star azimuths and elevations (in the local zenith frame), the shaft encoder readings, and the current autocollimation azimuth and elevation. The declination and hour angle of the star is also tabulated to facilitate cross-referencing the pointing data with the observing log, etc.

The original `susipoint` program used the simplex algorithm (J.A. Nelder & R. Meade (Computer Journal **7** 308, 1965) to minimize

$$\chi^2 = \sum |\hat{\mathbf{r}}_e - \hat{\mathbf{r}}_m|^2 \tag{5.1}$$

where $\hat{\mathbf{r}}_e$ is the observed star position as determined from the shaft encoder readings and $\hat{\mathbf{r}}_m$ is the predicted or model position.

The original `susipoint` code is rather hard to follow although it appears to be correct.

The new `susipoint` program again uses the Nelder and Meade simplex algorithm, but the function that is minimized is

$$\chi^2 = \sum [(a_m - a_e)^2 + ((\epsilon_m - \epsilon_e)^2] \tag{5.2}$$

where $(a_m,\ \epsilon_m)$ and $(a_e,\ \epsilon_e)$ are respectively the model and encoder azimuths and elevations. This is slightly simpler to implement than the original. In the limit of small angles the two criteria are essentially the same.

The new `susipoint` uses the `astromod` where possible. This guarantees that `susipoint` uses exactly the same algorithms as `astromod`. To facilitate this, the `susipoint` code is located in the `astromod` directory. The makefile includes the line

```
vpath %.c ../server
```

This directs `make` to search the directory `../server` for source code files and ensures that any changes to the model functions will be correctly updated in `susipoint`.

The only algorithmic problem that I encountered was the fact that the simplex scaling factor needs to be less than 1, otherwise the simplex search wanders into the land of NaN (not a number). A value of 0.5 seems to be OK.

The new `susipoint` recomputes the model using the tabulated autocollimation angles for each point in the file. This significantly slows the calculation somewhat compared to using a fixed autocollimation vector for all points.[4]

[4] I use a simple brute-force method of recomputing the mirror position for every data point. This increases the number of model recalculations by n, where n is the number of data points. One could speed things up by treating all the points with a given autocollimation vector together, but this complicates the bookkeeping. The brute force method is slow but doesn't require any tricksy code.

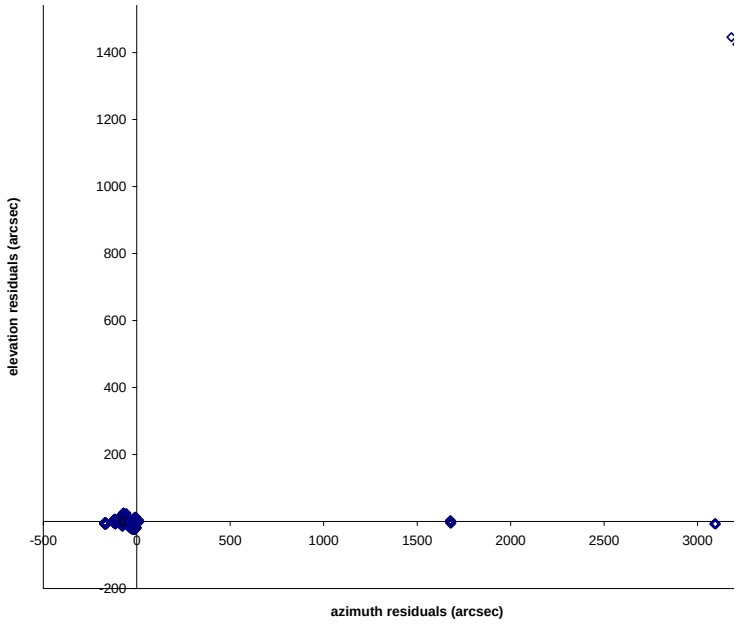

Figure 5.1: The azimuth and elevation residuals for S4 on the night of 2005 March 2. The errors are dominated by a handful of discrepant points.

5.1 Example I

The first test of the new `susipoint` was on a data set provided by Mike that he had analyzed using IDL. All the data were taken using the south periscope. The fitting parameters predicted by `susipoint` were quite different to Mike's results. In particular, I found a negative θ_1 and θ_2 close to zero. However, Mike had found that fitting for θ_2 was not very reliable, and he found the best fit by doing a grid search in θ_2. When I used his value of $\theta_2 = 214^\circ$ I got essentially the same results as Mike (his results are the current model parameters and are listed later in this section).

5.2 Example II

As mentioned previously, an "all sky" solution should be better at determining the various fitting parameters. This is something that could not be done with the original program.

As a second test I analyzed data taken on the night of 2 March 2005 using both north- and south-looking data. Mike produced a merged data file having 1139 data points.

It took 36 s to process this file with `susipoint` on `arthur`, and the program found 280 bad data points. The rms fitting errors in azimuth and elevations were $378''$ and $61''$. The χ^2 defined by Eq. (5.2) was 3.9×10^{-3}. These errors are fairly typical of what we have seen with the old program. The residuals are plotted in Fig. 5.1, and it is clear that the residuals are dominated by a handful of wildly discrepant points.

The 859 "good" points were analyzed again (15 s). No bad points were reported. The rms residuals in azimuth and elevation were respectively 6.99 and 5.65 arcsec. The improvement in the fit is remarkable: a factor of ~ 50 in azimuth and ~ 10 in elevation.

As with the south-looking data set, θ_1 is negative, suggesting that θ_2 should be $\sim 180°$. I did a crude grid search using fixed values of θ_2 and found that the best value was around $195°$. A final fit using this as a starting value for θ_2 gave the following results:

```
chi^2 = 1.51072e-06 after 893 iterations and 1358 models.
Model parameters in degrees (starred parameters are fixed):
acol az offset:     0.122193
acol el offset:     0.001730
delta 1:            0.389685
delta 2:            0.703845
theta 1:            0.011875
theta 2:          198.713849
az encoder offset: -0.140385
el encoder offset: -0.095932
Azimuth and elevation residuals (in arcsec):
RMS az & el errors (arcsec): 6.67 5.51
```

The parameter θ_1 is now positive as one would expect.

There is one problem with this solution: the azimuth autocollimation offset is rather large (approximately 7 arcmin). If the autocollimation offsets are set to zero we get

```
chi^2 = 1.67385e-06 after 334 iterations and 566 models.
Model parameters in degrees (starred parameters are fixed):
acol az offset:     0.000000*
acol el offset:     0.000000*
delta 1:            0.035300
delta 2:            0.124136
theta 1:            0.012582
theta 2:          196.035867
az encoder offset:  0.319721
el encoder offset: -0.092383
Azimuth and elevation residuals (in arcsec):
RMS az & el errors (arcsec): 7.10 5.70
```

As one would expect the fit is not quite as good as the previous one. The residuals for this solution are shown in Fig. 5.2. The filled points in the plot correspond to south-looking data; the open points were observed with the North periscope.

The current values of these parameters, taken from the `sidstats` file, are based on Mike's analysis of the south-looking data only:

```
delta1  -0.0244
delta2  -0.0420
theta1   0.0228
theta2 214.000
azoff    0.4706
eloff   -0.1016
```

Not surprisingly, there are differences between this set and the set found using all-sky pointing.

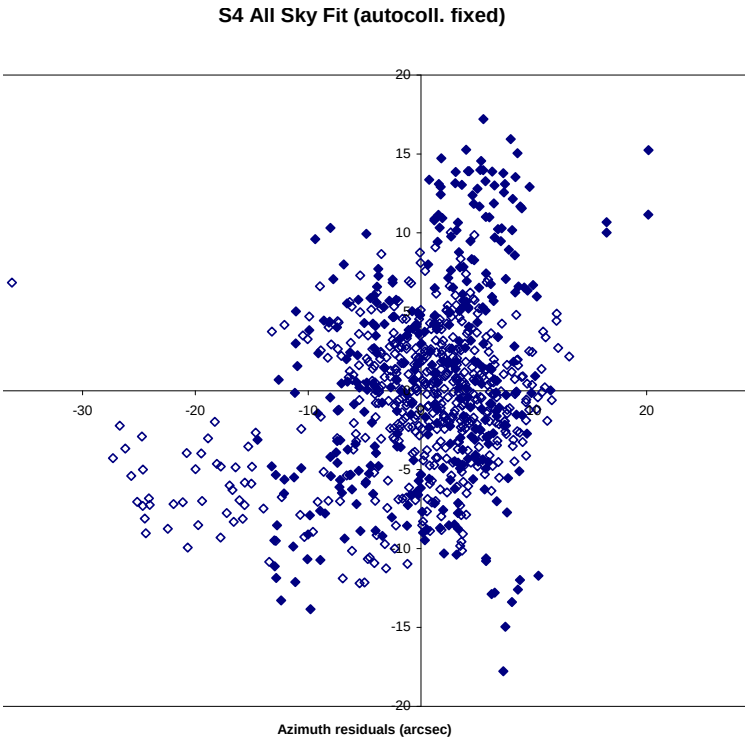

Figure 5.2: The final all-sky fit for S4 after optimizing the value of θ_2. The autocollimation offsets are fixed at zero. The open and filled points denote north- and south-looking data, respectively. Note that the elevation and azimuth scales are not the same.

Although these results look very encouraging, there are still some problems. The most worrying of these can be seen if we magnify the central part of Fig. 5.1. This region is shown in Fig. **??**. The azimuth residuals do not have a Gaussian distribution. They in fact lie in four distinct groups, located roughly at -15, -75, -115 and -170 arcsec. The last two groups have been marked as "bad" by `susipoint` and are among the 280 points that were rejected. The group at -15 arcsec consists mainly of data taken with the North periscope; the -75 arcsec group corresponds to south-looking observations.

Surprisingly, the two groups apparently merge in the "cleaned" fit shown in Fig. 5.2. One possible explanation is that the fitting procedure has used the offset in the azimuth autocollimation position to minimize the difference between the two groups. I thought that this might be a side-effect of allowing the program to fiddle with the autocollimation angles, but this in the end turned out to be a red herring.

I found the explanation of these weird effects using a "feature" of `susipoint` originally included to prevent the program crashing when the user did silly things. My implementation of the simplex algorithm, originally written for binary star orbits, allows one to fix some of the fitting parameters. My concern was what would happen if *all* the parameters were fixed? Normally one would never do this, but I didn't want the algorithm to fall over simply because a user accidently gave it silly numbers. The simplex routine in this case simply calculates the χ^2 for the data set, using the fixed parameters, and returns without

doing any optimization.

I realized that this feature could actually be useful with `susipoint`. If we run the *original data set*, dodgy points and all, using the final fitting parameters as fixed quantities, it would show up any strange effects.

The explanation for the "bunching" of points is now clear: the original fit is badly affected by the outlier points. The siderostat model is skewed and the residuals will vary with azimuth and elevation across the sky. In fact, some "good" data points can have large residuals that cause them to be rejected.

I suggest the following procedure:

1. Run `susipoint` on the original data set.
2. Use the "cleaned" data in the output file as a new input file for the fitting program. For the reasons just discussed, some of the rejected data points may in fact be OK. Don't worry. Be happy.
3. You may need to manually tweak θ_2, and it is probably a good idea to run solutions with and without the autocollimation offsets included.
4. Run `susipoint` again on the original data set, but this time *fix all the parameters* to what you think is the best fit. If the number of rejected points is unchanged, then the fit is OK. If the number of rejected points is much less this time, use the cleaned data in the output file as the input to a new fitting cycle.
5. Use common sense and monitor the process by plotting the residuals at each stage.

Following this recipe for the S4 data, I obtained the following results:

```
chi^2 = 1.95e-06 (2.78e-06) after 650 (389) iterations.
Model parameters in degrees (starred parameters are fixed):
acol az offset:     0.083056   0.000000*
acol el offset:     0.006818   0.000000*
delta 1:            0.248869   0.002001
delta 2:            0.441513   0.026856
theta 1:            0.013758   0.013016
theta 2:          190.728022 190.107043
az encoder offset:  0.077537   0.416863
el encoder offset: -0.098449  -0.090953
Azimuth and elevation residuals (in arcsec):
RMS az & el errors (arcsec): 6.22 6.02 (8.43 5.97)
```

where the second set of numbers refers to a fit with the autocollimation offsets fixed to be zero. A total of 1108 points were included in the fit and no data were rejected. The residuals are plotted in Fig. 5.3. Although this looks quite similar to Fig. 5.2 it includes more than 200 additional observations. *Only 31 outliers have been rejected.*

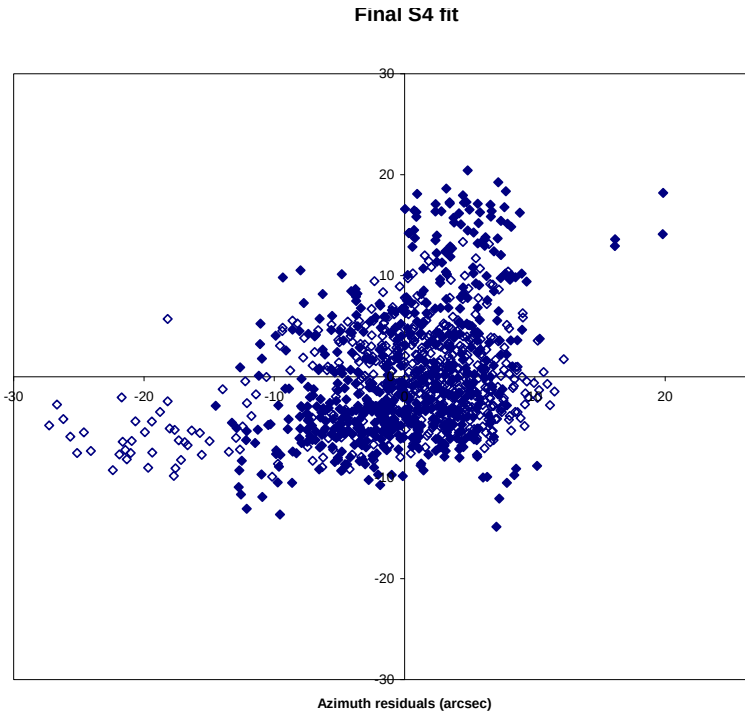

Figure 5.3: The final fit to the data set, omitting only the 31 outliers. The autocollimation offsets have been included in the fit. The filled points are south-looking, the open points are north-looking.

6 Conclusions

It seems clear that the main reason for poor pointing solutions has resulted from the contamination of the pointing data by outliers. In the Introduction we discussed a number of possible reasons for the outliers, but the results presented in section 5 suggests that none of these are a major problem. The real reason appears to be that `susipoint` may converge on the wrong solution, artificially creating a large number of outliers. Most non-linear least squares algorithms use a variation of the method of steepest descent; i.e., they are based on the hypothesis that the true minimum can be found by following the path that maximizes the gradient. Steepest descent methods, however, fail when there are many "valleys" and local minima. The simplex algorithm used by `susipoint` has heuristic rules that in general work well, but it can also converge to the wrong minimum. Grid searching over the entire parameter space is feasible but very slow.

The iterative procedure described in section 5 appears to work well. I have modified `susipoint` to make it easier to process the data iteratively, and the details are given in the companion document *How to use "susipoint" to find siderostat pointing solutions.*

A The local zenith frame

The local zenith frame of reference is defined by the three orthogonal unit vectors $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$ and $\hat{\mathbf{z}}$. The vector $\hat{\mathbf{z}}$ points to the local zenith, $\hat{\mathbf{x}}$ lies in the horizontal plane and points East, and $\hat{\mathbf{y}}$ also lies in the horizontal plane and points North.

Let $\hat{\mathbf{r}}$ be a unit vector pointing in some arbitrary direction. Its cartesian components are

$$\hat{\mathbf{r}} = r_x\hat{\mathbf{x}} + r_y\hat{\mathbf{y}} + r_z\hat{\mathbf{z}} \tag{A.1}$$

where

$$\hat{\mathbf{r}}_x = \hat{\mathbf{r}} \cdot \hat{\mathbf{x}} \tag{A.2}$$

etc. The components r_x, etc., are often called direction cosines, particularly in the older literature.

Because $\hat{\mathbf{r}}$ is a *unit* vector the three direction cosines are not independent, since $r_x^2 + r_y^2 + r_z^2 = 1$. Only two independent parameters are required to uniquely specify $\hat{\mathbf{r}}$. In particular, $\hat{\mathbf{r}}$ can be specified by two angles. There are several choices of angles depending on the coordinate system being used and the application at hand. In the local zenith frame the most commonly used angles are the azimuth a and elevation ϵ. The azimuth is measured from the y-axis and increases in a clockwise sense, so $a = 90^\circ$ corresponds to the x-axis. The elevation is measured from the horizontal plane and increases in a counterclockwise direction. The zenith is at an elevation $\epsilon = 90^\circ$. The zenith distance $\zeta = \pi/2 - \epsilon$ is often used. It has the advantage that, like the azimuth, it increases in a clockwise direction from 0 at the zenith to $90\circ$ on the horizon. Writing $\hat{\mathbf{r}}$ explicitly, we have

$$\hat{\mathbf{r}} = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} = \begin{bmatrix} \cos\epsilon \sin a \\ \cos\epsilon \cos a \\ \sin\epsilon \end{bmatrix} = \begin{bmatrix} -\cos\delta \sin h \\ \cos\phi \sin\delta - \sin\phi \cos\delta \cos h \\ \sin\phi \sin\delta + \cos\phi \cos\delta \cos h \end{bmatrix} \tag{A.3}$$

The last vector in this equation expresses the direction cosines in terms of the local equatorial frame, where h is the hour angle, δ the declination, and ϕ is the latitude of the observer. The hour angle and declination are another pair of angles that can be used to uniquely specify $\hat{\mathbf{r}}$. Another set of angles is the right ascension α and the declination δ. These give the coordinates of $\hat{\mathbf{r}}$ in a frame of reference that is fixed with respect to the celestial sphere. Of course, because of the Earth's rotation this frame appears to rotate with a period of one sidereal day. The hour angle and right ascension are related through the formula $h = t - \alpha$ where t is the local apparent sidereal time.

B Modelling the shaft encoders

Each siderostat axis actually has *two* shaft encoders. One encoder is "absolute:" its output is a digital signal in the range 0–359 and accurately reports the shaft position in steps of one degree.[5] The elevation encoders are directly connected to the elevation shafts. The azimuth encoders, for practical reasons, are connected to the azimuth shaft through a gear train having a gear ratio of 1:1.

[5]When these were purchased in the late 1980s they were regarded as "optical munitions" and required an export license issued by the US Department of Defense!

The elevation and azimuth motors drive the shafts through 360:1 worm gears. The gears were manufactured by Edward Byers Engineering of Barstow, CA. Until Ed Byers retirement, his gears were widely regarded as the most precise commercially available gears designed specifically for astronomical applications.

"Incremental" encoders are attached to each worm. Incremental encoders have three outputs.

- The "A" output is a rectangular waveform having a frequency Nf, where N is the number of "lines" ruled on the encoder and f is the motor rotation rate. Price is a strong function of N! The SUSI encoders have $N = 1024$ and a motor speed of one revolution per second would consequently generate a square wave having the frequency 1024 Hz (this is actually the maximum slew speed of the siderostats).

- The "B" output is similar to the "A" output except that it is shifted by $90°$. The direction of rotation can be determined by noting which of the two phases is leading. It is also possible to use the quadrature signal to improve the precision by a factor of two; to date this hasn't been done with SUSI.

- The "index pulse" is a short pulse that occurs once per revolution. It is usually connected to a counter to keep track of the whole number of turns made by the encoder.

Incremental encoders are much cheaper than absolute encoders. Their main disadvantage is that they only measure relative motion. If there is a power failure, etc., they need to be reset by returning to a fixed fiducial. In SUSI the the zero point of the incremental encoders is determined by the indexing pulse. The absolute encoder is read at the same time to determine the whole number of degrees. The shafts never have to be driven by more than $1°$ to "index" the incremental encoders.

Since the main worm gear has a ratio of 360:1, a single step of the incremental encoder corresponds to approximately $0.^{\circ}001$ or ~ 3 arcsec.

The axis position is determined by a combination of hardware electronics and software in the AV68K. The hardware determines the motor direction (from the relative phase of the A and B waveforms) and the incremental encoder "ticks" are counted by an up/down counter. This counter is reset to zero by the indexing pulse.

The actual algorithm for determining the position from the incremental and absolute encoders is surprisingly complicated. It has to work properly in both directions (clockwise and counterclockwise rotations) and gear backlash is a problem. We still get occasional errors that require re-indexing.

In my view these errors are most likely due to electrical noise on the index pulse line. If an erroneous reading occurs while acquiring a star the siderostat will be incorrectly positioned and it is unlikely that the star will be found. It is usually quicker to re-index the siderostat and reacquire the star rather than to try and search for the star. If the error occurs during a slew the software may lose control of the siderostat.

If an error occurs while tracking it is usually benign. However, if pointing data is being collected the bad data will bias the pointing solution.

The incremental and absolute encoders have to be set mechanically and the indexing pulse position must be located approximately midway between the degree transitions on the absolute encoder. Because of this, and because of the inaccessibility of some of the encoders

(the azimuth encoders can only be accessed with great difficulty) it is not very practical to adjust them to make "0.0000" exactly horizontal or due North. As a result the encoders will have a constant offset. Using θ to denote either azimuth or elevation as appropriate, the true siderostat angle can be found from

$$\theta_s = \theta_e - \Delta\theta \tag{B.1}$$

where the subscript e denotes the encoder reading. The offsets can be as much as one or two degrees.

Because the incremental encoder is rigidly connected to the worm drive, any periodic gear error will cause the incremental encoder reading to be in error. The periodic term can be modelled by

$$\theta_p \cos(2\pi\theta_i/1024 + \phi)$$

where θ_p is the amplitude of the variation (in encoder units), θ_i is the incremental encoder reading, and ϕ is a phase.

One can obviously generalize this procedure to include other periodic terms.

B.1 The encoder algorithm

The AV68K routines that determine the shaft position is called `SynAXEncoder`. To simplify code maintenance the azimuth and elevation control code are based on a common set of functions. If the preprocessor variable `AZIMUTH` is defined the various "`AX`" definitions are resolved to refer to the azimuth axis, otherwise they refer to the elevation axis. The "`D_REGxxx`" registers are copies of hardware registers in the station control electronics.

```
#ifdef AZIMUTH
    void SyncAzEncoder()
#else
    void SyncElEncoder()
#endif

{
    /* Internals */
    int current_value;
    int index;
    int direction;


    /*  Don't do anything if AX hasn't been indexed!    */
    if (AX_DEGR == 999)
        return;

    /*
     *  Get current values of the encoders,
     *  the state of the index pulse and the direction of
     *  motion. If the indexing
     *  pulse is present set the hysteresis bit in the
     *  SID_FLGS status register.
     */

    AX_FRAC = IncrEncoder(READ_REG_AX);
    current_value = BCD2Num(AbsEncoder(READ_REG3));
    direction = D_REG_RS & AX_DIR;

    if ( !(D_REG_RS & AX_IDX) )
    {
        index = TRUE;
        SID_FLGS |= AX_HYSTERESIS;
    }
    else
        index = FALSE;

    /*
     *  Determine the state of the encoders and
     *  set AX_DEGR (the prevailing value of the
     *  absolute encoder) accordingly. There are
```

```
     *  four (exclusive) possibilities:
     */

    /*
     *  I.
     *  First see if we are at zero.  If so, AX_DEGR equals
     *  the current value of the encoder.  There is a sublety
     *  here!  We must check whether the incremental encoder
     *  is reading zero, NOT whether the index pulse is present.
     *  The pulse can disappear before the incremental encoder
     *  ticks over to a non-zero count.  If we rely solely on
     *      the index pulse we can get a one degree error for a
     *  short while while traveling in the -ve direction. We
     *  should only change AX_DEGR if the hysteresis bit is
     *  set.  This indicates that the indexing pulse was indeed
     *  detected in an earlier cycle. If we don't check this
     *  there is a possibility that the incremental encoder
     *  could move to (say) 1023 and "bounce" back to zero
     *  without setting the index flag.  Sounds farfetched but
     *  it might work...
     */
    if (!AX_FRAC && (SID_FLGS & AX_HYSTERESIS) )
    {
        AX_DEGR = current_value;
    }

    /*
     * II.
     * Index is still present, but incremental encoder is non-zero.
     * This will occur if the motor is moving fairly rapidly.  The
     * index pulse was latched during the previous clock cycle and
     * the shaft has moved between then and the beginning of the
     * current cycle. Use the "prevailing" value of the absolute
     * encoder, i.e., the actual reading for +ve motion and
     * (actual - 1) for -ve direction. Don't clear the hysteresis
     * bit just yet.
     */
    else if (index && AX_FRAC)
    {
        AX_DEGR = current_value;
        if (direction)
        {
            if (AX_DEGR )
                AX_DEGR--;
            else
                AX_DEGR = 359;
        }
    }

    /*
     * III.
     * We have just moved away from indexing point. The AX_DEGR
     * value is calculated as above but now we clear the hysteresis
     * bit. This will turn off any recalculation of AX_DEGR until
     * the next indexing point is encountered.
     */
    else if (!index && (SID_FLGS & AX_HYSTERESIS) )
    {
        AX_DEGR = current_value;
        if (direction)
        {
            if (AX_DEGR)
                AX_DEGR--;
            else
                AX_DEGR = 359;
        }
        SID_FLGS &= ~AX_HYSTERESIS;
    }

    /*
     * IV.
     * AX_DEGR is now set.  We got here either from one of
     * the above tests (I-III) or there was no index pulse and
     * no hysteresis flag. In all cases there is nothing more
     * to do.
     */
    return;

}/* SyncAXEncoder */
```

C Brief summary of the model functions

The siderostat pointing model is defined by functions in a file called `sidmodel.c`. A brief synopsis of these functions is given here.

C.1 InitializeModel(*north, *south)

The arguments of this function are pointers of type `SIDEROSTAT` (defined in the standard SUSI header `susilib.h`). The `SIDEROSTAT` structure contains a variety of data relating to the given siderostat.

This function allocates storage for all the rotation matrices. It then calls the function `ChangeModel`.

Two sets of matrices are created, one for each of the two siderostats in use. In `susipoint` the two arguments point to the same structure.

C.2 ChangeModel(*north, *south)

The `ChangeModel` function actually populates the various rotation matrices. It takes the model parameters from the two `SIDEROSTAT` structures pointed to by the arguments. The various trigonometric functions and shaft encoder offsets are copied to static global variables.

Depending on the siderostat orientation (north- or south-looking, specified by the `pscope` structure member), the appropriate autocollimation positions, expressed as shaft encoder readings, are converted to a unit vector in the local zenith frame of reference using the `azel2xyz` function. This vector is saved as part of the `SIDEROSTAT` structure.

In `susipoint` the two arguments point to the same structure and the autocollimation data is always put in the "north" positions.

The reason for splitting the initialization and setup functions in this way was to facilitate finding pointing solutions. The `ChangeModel` function can be called at any time to recompute the various matrices, etc.

C.3 azel2xyz(ns, az, el, *vector)

This function converts shaft encoder azimuth and elevation readings to a unit vector in the local zenith frame. The first parameter indicates which set of siderostat parameters (North or South) is to be used.

C.4 xyz2azel(ns, *mirror, *az, *el)

This is essentially the inverse of the previous function. Given the mirror vector coordinates in the local zenith frame, it returns the predicted shaft encoder readings.

C.5 MirrorPosRate(ns, *az, *el, *azrate, *elrate, *star)

This function computes the mirror position using Eq. (3.1). The star coordinates are input as a unit vector. The mirror azimuth and elevation are then calculated using the `xyz2azel` function.

The `astromod` function calculates positions and rates at fixed time intervals determined by the variable `TICK_TIME`. The rates are found numerically, using the previously calculated values of azimuth and elevation and `TICK_TIME`. In the present context the rates are irrelevant.