GBT MEMO 175

GBT Subreflector Actuator Functions in C

Don Wells*

January 21, 1998

Abstract

Two ANSI-C functions are described: `srDisplacementToLength()`, which accepts displacements (three linear, three angular) from the "home" position of the subreflector and computes six actuator lengths, and `srLengthToDisplacement()`, which performs an iterative numerical inversion of function `srDisplacementToLength()` (i.e., it accepts actuator lengths and produces displacements). Partial derivatives of the actuator lengths *wrt* translation and tilt are tabulated.

Contents

1 Introduction 2

1.1 Coordinate systems and axis permutations 2

2 The functions and their algorithms 4

2.1 The inverse kinematic problem: `srDisplacementToLength()` 4

2.2 The forward kinematic problem: `srLengthToDisplacement.c` 6

2.3 Actuator length calibration: `srActuatorsLengths.c` 8

3 Test cases & timing measurements 10

4 Partials of translation/tilt motions *wrt* actuator motions 10

5 Safety Considerations 15

A `srInclude.h` 16

Bibliography 18

*mailto:dwells@nrao.edu

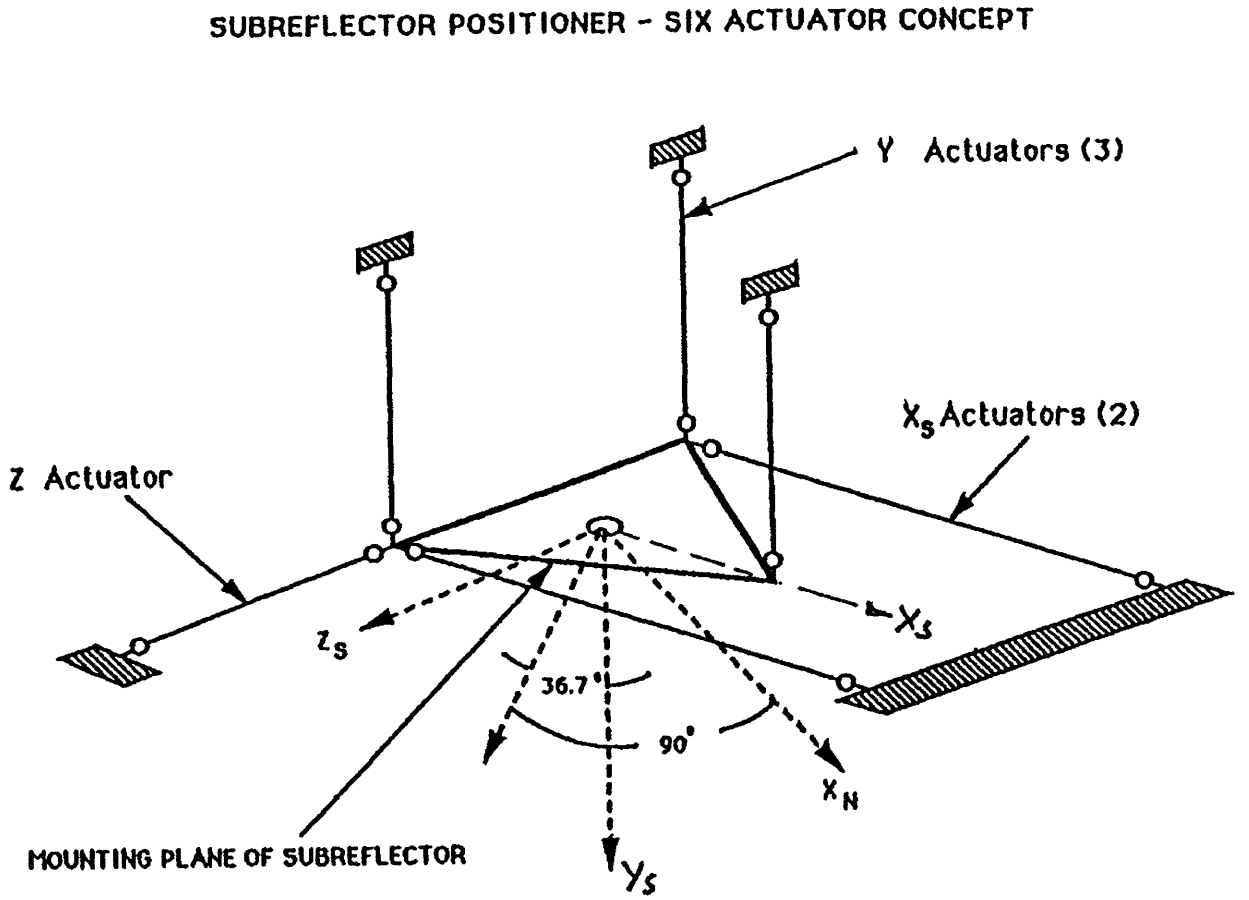

Figure 1: Conceptual drawing of the GBT 6-DOF Stewart platform

1 Introduction

The GBT subreflector actuator system is an example of a "Stewart Platform" [Ste66]. Such devices consist of a movable platform which is connected to a fixed structure by a set of linear actuators via U-joints. A wide variety of arrangements of the actuators and U-joint locations is possible. Figure 1 is a conceptual drawing of the arrangement used in the GBT subreflector. By studying the figure the reader can become convinced that the six linear actuators will implement three translation and three rotation motions, a total of six degrees of freedom. Figure 2 shows a view of the tip of the GBT feedarm as seen from the right side. The actuators are labelled, and their length ranges are noted.

1.1 Coordinate systems and axis permutations

The (x_s,y_s,z_s) subreflector axes shown in Figure 1 are *not* the same as the elevation (tipping) structure coordinate system which was discussed in [WK95]! The axes are permuted and rotated. The z_s axis corresponds to x_e. The y_s and x_s axes correspond (roughly) to z_e and y_e respectively (they are rotated by 36.7° about z_s/x_e). The origin of (x_s,y_s,z_s) shown in Figure 1 is at (0,-168.98,+2511.92) in the "reflector" coordinate system [KM93, Kin94], or (0,-2327.96,+4608.71) in the elevation coordinate system [WK95, pp.15–16], which is the subreflector "vertex". This point is *not* the mathematical vertex of the ellipsoid; rather it is the point of intersection of the axis of the feedhorn with the ellipsoid.

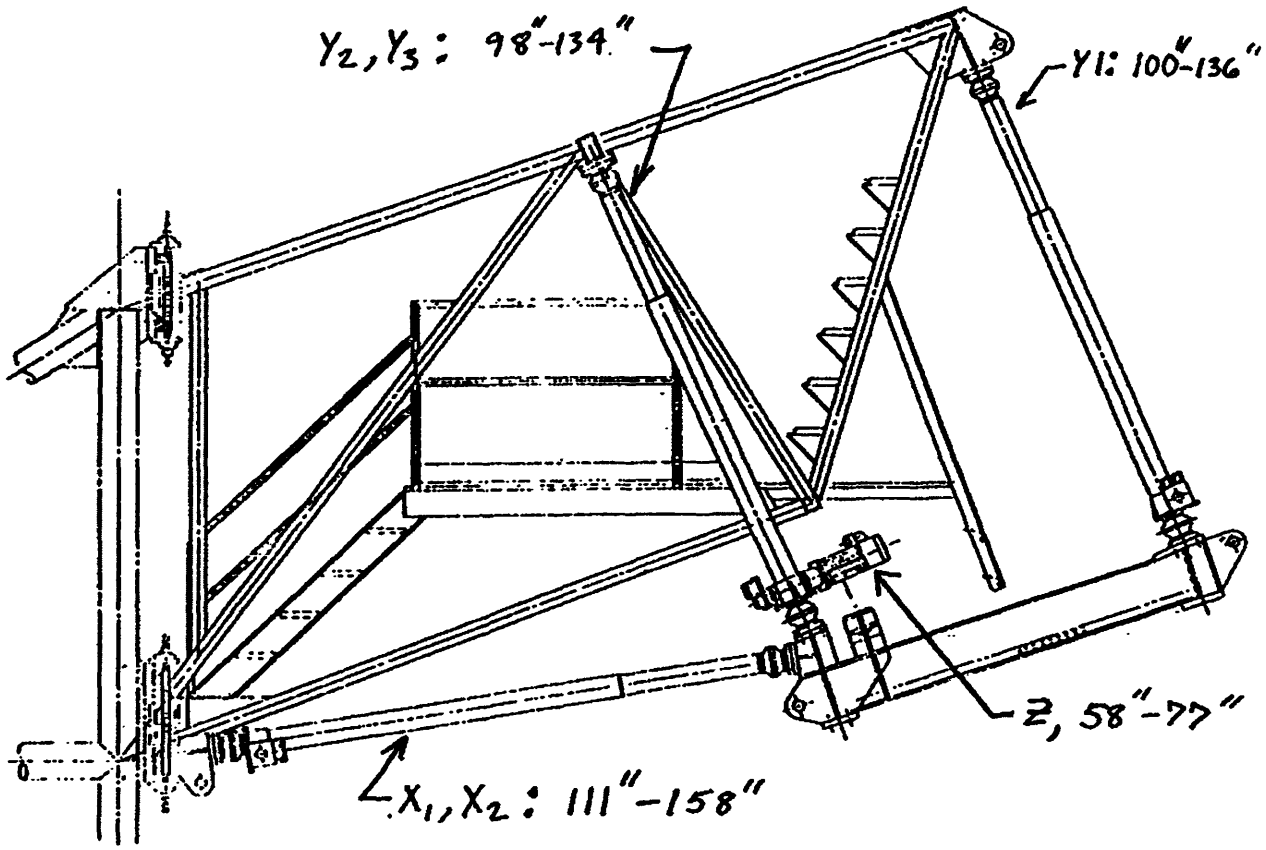

Figure 2: Side view drawing of the subreflector actuators (facing left)

Figure 1 also shows two other axes, "x_n" and an unlabelled axis 90° from it and 36.7° from y_s. The unlabelled axis is parallel to the axis of the paraboloid, z_r and z_e. x_n is parallel to y_e and y_r. The "n" of x_n is for "nutation", because it is the axis about which the subreflector should be rotated to move the beam in azimuth (constant elevation) between feedhorns. The 36.7° tilt of the subreflector coordinate system was specified by Srikanth [Sri90] so that motion of the subreflector in y_s *only* will move the Gregorian focus along the axis of the feedhorn.

Note that the axes of the (x_n,y_s,z_s) system are *skewed*: x_n is *not* orthogonal to y_s!
Also, we translate the subreflector along x_s, but we tilt it about x_n!

				platform U-joints		
encoder[6]	M&C index	servo amplifier name	Location	x_s (in)	y_s (in)	z_s (in)
0	1	Y1	tip of triangle	57.225	39.219	0.000
1	2	Y2	right-hand	-30.150	55.703	51.123
2	3	Y3	left-hand	-30.150	55.703	-51.123
3	4	X1	right-hand	-40.493	48.375	51.123
4	5	X2	left-hand	-40.493	48.375	-51.123
5	6	Z1	lateral motion	-18.495	48.856	45.473

Table 1: Identifications of the GBT subreflector actuators

2 The functions and their algorithms

The "forward kinematics problem" (given lengths L_i, find the platform pose) is analytically intractable. The inverse kinematics problem (given the pose, find L_i) is, however, easy to compute: transform the coordinates of the platform U-joints using the pose parameters and then the lengths are obtained as the Euclidian distances between the respective U-joints. The forward problem can then be solved numerically by iterative inversion of the inverse algorithm. The GBT inverse and forward kinematic problems have been implemented in C, as described below.

The original version of the code below was constructed during April and May of 1993 by the author. It was all in one file named `tm19.c` (the name refers to [Nan91]), which was delivered to the Precision Controls Division [PCD] of CRSI in May 1993. PCD incorporated the code into the GBT's Operator Control Unit [OCU]. The `tm19.c` version of these algorithms did not include actuator end-point offsets, because the OCU has no need for this refinement.

Fred Schwab (`fschwab@nrao.edu`) has provided advice on the mathematical methods used in these implementations, especially for `srLengthToDisplacement()`.

2.1 The inverse kinematic problem: `srDisplacementToLength()`

```
int srDisplacementToLength(                    /* returns !=0 on delta-errors  */
                           double trans[3],    /* delta translations      (in) */
                           double tilts[3],    /* delta tilts           (deg) */
                           double temp,        /* temperature         (deg_C) */
                           double encoder[6], /* output: lengths to use  (in) */
                           double apex_delta[3][6],     /* U-joint errors(in) */
                           double home_sub_delta[3][6]) /* U-joint errors(in) */
```

Function `srDisplacementToLength()` is a transliteration from Basic to ANSI-C of the algorithm given in [Nan91]. The original version accepted only two angular displacements. `srDisplacementToLength()` supports a third angle, θ_y, by inserting another rotation matrix after the θ_x rotation (if $\theta_y = 0$, this is an identity matrix, consistent with the algorithm of [Nan91]). Rotation about θ_y is needed for focus tracking, because of the elevator asymmetry.

```
/* srDisplacementToLength.c --- compute actuator lengths from displacements
   D.Wells, NRAO-CV
   1993-05: original version
   1995-05: this function now in separate file
   1996-02: changes
   1997-04-02: change disp[6] argument to trans[3] & tilts[3].
   1997-10-28: changes to implement new mathVectorMatrix macros
   1997-11-10: swap order of rotations: (Y,-ty) now is after (Z,+c1)
   */
```

[GNU GPL copyright notice omitted]

```
#include "srInclude.h"
#include "mathVectorMatrix.h"
#include "gbtOpticalConstants.h"

int srDisplacementToLength(                /* returns !=0 on delta-errors  */
                           double trans[3], /* delta translations      (in) */
                           double tilts[3], /* delta tilts           (deg) */
                           double temp,     /* temperature         (deg_C) */
```

```
                        double encoder[6], /* output: lengths to use  (in) */
                        double apex_delta[3][6],     /* U-joint errors(in) */
                        double home_sub_delta[3][6]) /* U-joint errors(in) */
{
  int m, i, j, status;
  double
      c1,    /* subreflector coor system rotation */
      tx,    /* tilt about X-axis */
      ty,    /* tilt about Y-axis */
      tz,    /* tilt about Z-axis */
      ma[3][3],
      sum, lsum, diff,
      new_sub_end[3][6],
      actual[6];       /* desired actual lengths of the actuators */
  MVM_PRIVATE_VARIABLES
```

The following two arrays of U-joint coordinates of the (fixed) feedarm apex ends of the 6 actuators and the "home" position of the subreflector ends of the actuators are from [Sch92, p.4], and are identical to those in [Zai92, p.A9]. They are in the subreflector coordinate system (rotated 36.7° *wrt* the tipping and the reflector coordinate systems) and are in inches.

```
  const double apex_end[3][6] = {
    /*  L1       L2       L3        L4       L5       L6            */
    { 58.391, -29.753, -29.753, -163.08, -163.08,  -17.798}, /* X_s */
    {150.161, 166.683, 166.683,  82.478,  82.478,  70.290}, /* Y_s */
    {  0.0,    51.123, -51.123,  51.123, -51.123, -12.25 }  /* Z_s */
  };
  const double home_sub_end[3][6] = {
    /*  L1       L2       L3        L4       L5       L6            */
    {57.225,  -30.15,  -30.15,   -40.493, -40.493, -18.495}, /* X_s */
    {39.219,   55.703,  55.703,   48.375,  48.375,  48.856}, /* Y_s */
    { 0.0,     51.123, -51.123,  51.123, -51.123,  45.473}  /* Z_s */
  };
```

Function arguments `home_sub_delta[][]` and `apex_delta[][]` specify offsets to the above endpoint coordinates; these arrays must be defined by the code which calls function `srDisplacementToLength()`.

Step 1 Compute rotation matrix to transform endpoints

This operation is done as five rotations:[1]

- Transform from x_s to x_n axis (36.7° rotation)
- Rotation θ_x about x_n axis
- Transform from x_n back to x_s axis
- Rotation θ_y about y_s axis[2]
- Rotation θ_z about z_s axis

```
  c1 =  GBT_SR_AXIS_TLT * DTR;  /* 36.7 deg in gbtOpticalConstants.h */
  tx = tilts[0] * DTR;          /* DTR=degrees-to-radians in <ditto> */
  ty = tilts[1] * DTR;
  tz = tilts[2] * DTR;
  ANGLES_2_MATRIX(MC, ma, Z,-c1, X,+tx, Z,+c1, Y,-ty, Z,+tz);
```

[1] Macro `ANGLES_2_MATRIX()`, which is invoked in this step, is defined in `mathVectorMatrix.h`; it computes a rotation matrix from up to five rotations about specified axes.

[2] This rotation was not included in [Nan91]. Also, in earlier versions of this function, such as the code shipped to PCD in May 1993 and incorporated by them into the OCU software, this rotation by θ_y was done in the transformed (x_n) coordinate system rather than in subreflector coordinates.

Step 2 **Compute the new subreflector ends of the actuators**

Rotation matrix `ma[][]` is multiplied by vector `home_sub_end[]` to get the new lengths. The empirical calibration matricies `hom_sub_delta[][]` and `apex_delta[][]` are added to the design values of the end-point coordinates.

```
for (m = 0; m < 6; m++) {
  for (i = 0, lsum = 0.0; i < 3; i++) {
    for (j = 0, sum = 0.0; j < 3; j++)
      sum += (ma[i][j] * (home_sub_end[j][m] + home_sub_delta[j][m]));
    new_sub_end[i][m] = trans[i] + sum;
    diff = (new_sub_end[i][m] - (apex_end[i][m] + apex_delta[i][m]));
    lsum += (diff * diff);
  }
  actual[m] = sqrt (lsum);
}
```

Step 3 **Compute actuator encoder values to use**

We must correct for actuator encoder zero point and scale errors, and for the effective thermal expansion coefficient of the actuator. See discussion in section 2.3.

```
for (m = 0; m < 6; m++)
    encoder[m] = srActuatorsLengths(m, actual[m], temp);
```

Step 4 **Test the maximum-delta rules**

These rules were specified by the Loral engineers who designed the GBT Gregorian subreflector actuators; they assert [Zai92] that no binding or interference of the mechanism will occur if these rules are obeyed, and if positions stay within the "required motion" box of the GBT.

```
  status = 0;
  if (fabs(encoder[0] - encoder[1]) >  2.60) status |=  1;
  if (fabs(encoder[0] - encoder[2]) >  2.60) status |=  2;
  if (fabs(encoder[1] - encoder[2]) >  2.62) status |=  4;
  if (fabs(encoder[3] - encoder[4]) >  1.40) status |=  8;
  if (    (encoder[1] - encoder[5]) < 39.19) status |= 16;
  if (    (encoder[1] - encoder[5]) > 62.98) status |= 32;

  return (status);
}
```

2.2 The forward kinematic problem: `srLengthToDisplacement.c`

```
int srLengthToDisplacement(                    /* returns !=0 if maxdelta fails */
                           double len[6],      /* input actuator lengths   (in) */
                           double temp,        /* temperature           (deg_C) */
                           double trans[3],    /* delta translations       (in) */
                           double tilts[3],    /* delta tilts             (deg) */
                           double apex_delta[3][6],      /* U-joint error(in) */
                           double home_sub_delta[3][6],  /* U-joint error(in) */
                           int    *iter)                 /* N-R iterations    */
```

The algorithm of `srLengthToDisplacement()` is the Newton-Raphson iterative method for solving a system of nonlinear equations. N-R multiplies the inverse of the Jacobian (partial derivative) matrix by the residual

vector to obtain a vector of corrections which are subtracted from the current solution vector. In a nonlinear problem, such as this one, the Jacobian matrix varies with the solution vector. Because the inverse Jacobian is expensive to compute we approximate it by using only the inverse Jacobian at the "home position" of the subreflector; this works because this problem is nearly linear in the neighborhood of the home position. The entire range of valid lengths of the actuators has not been explored to verify that the inversion works for all cases, especially when argument `disp[]` is initialized to zero (intermediate iterations might violate the length rules). Normally we will have `disp[]` $= \epsilon$, and the problem is nearly linear; use of an approximate inverse Jacobian should only result in extra iterations in extreme cases.

```
/* srLengthToDisplacement.c --- compute displacements from  actuator lengths
   D.Wells, NRAO-CV
   1995-05/96-02
   1997-04-02: change disp[6] argument to trans[3] & tilts[3].
   1997-11-10: minor change of comments
*/
```

[GNU GPL copyright notice omitted]

```
#include <math.h>
#include "gbtOpticalConstants.h"
#include "srInclude.h"

int srLengthToDisplacement(                 /* returns !=0 if maxdelta fails */
                           double len[6],   /* input actuator lengths   (in) */
                           double temp,     /* temperature           (deg_C) */
                           double trans[3], /* delta translations       (in) */
                           double tilts[3], /* delta tilts            (deg) */
                           double apex_delta[3][6],     /* U-joint error(in) */
                           double home_sub_delta[3][6], /* U-joint error(in) */
                           int    *iter)                /* N-R iterations    */
{
```

This function iterates the Newton-Raphson algorithm until the computed lengths agree with the input lengths. It uses `trans[]` and `tilts[]` as its initial value for the iterations (i.e., `trans[]` and `tilts[]` are both input and output arguments of the function). The calling program should either initialize `trans[]` and `tilts[]` to zeroes, or should use the last returned pose if the new lengths are almost the same as for the last call.

```
    int i, j, status;
    double lenp[6], resid[6], adiff, biggest_resid, sum, v[6];
    const int maxiter = (10);
    const double tolerance = (0.0001); /* inches */
```

The following Inverse Jacobian matrix `ji[6][6]` of the transformation (see [Sch92, Eq.(4)]) was computed at the "home" position using a Mathematica program supplied by F.Schwab. The values are essentially identical to those of Eq.(11) on p.6 of [Sch92], except that the signs of the partials of length *wrt* angle in rows 4–6 are flipped because of an opposite convention for angle sense.

```
    const double ji[6][6] = {
      /*   L1          L2          L3          L4          L5          L6          */
      {-0.5202510,+0.1216542,+0.1216542,+0.5166016,+0.5166016, 0.0      },/*dX*/
      {-0.3450957,-0.3266342,-0.3266342,-0.0030950,-0.0030950, 0.0      },/*dY*/
      {-0.0491685,-0.7560834,+0.4310301,-0.1789911,+0.1897813,+1.0667840},/*dZ*/
      { 0.0      ,+0.0094583,-0.0094583,-0.0060318,+0.0060318, 0.0      },/*tX*/
      { 0.0      ,+0.0036598,-0.0036598,+0.0081530,-0.0081530, 0.0      },/*tY*/
```

```
    {-0.0114321,+0.0057268,+0.0057268,-0.0000410,-0.0000410, 0.0      } /*tZ*/
  };

  for (*iter = 0; *iter < maxiter; (*iter)++) {
```

Step 1 **Compute actuator lengths**

```
    status = srDisplacementToLength (trans, tilts, temp, lenp,
                                     apex_delta, home_sub_delta);
    if (status != 0) break;
```

Step 2 **Compare computed lengths to input lengths**

```
    biggest_resid = 0.0;
    for (i = 0; i < 6; i++) {
        resid[i] = (len[i] - lenp[i]);
        adiff = fabs (resid[i]);
        if (adiff > biggest_resid) biggest_resid = adiff;
    }
```

Step 3 **Convergence test**

Terminate iteration loop if biggest length residual is less than the tolerance.

```
    if (biggest_resid < tolerance) break;
```

Step 4 **otherwise, compute new correction vector**

The residual vector `resid[]` is multiplied by the matrix J^{-1} to get the Newton-Raphson correction vector `v[]`:

```
    for (j = 0; j < 6; j++) {
        for (sum = 0.0, i = 0; i < 6; i++) sum += (resid[i] * ji[j][i]);
        v[j] = sum;
    }
```

Step 5 **Finally, apply the correction vector (update the pose vectors)**

```
    for (i = 0; i < 3; i++) {
        trans[i] += v[i];
        tilts[i] += (v[i+3] / DTR);
    }
  }
  return (status);
}
```

2.3 Actuator length calibration: `srActuatorsLengths.c`

```
double srActuatorsLengths(              /* returns act j encoder length (in) */
                          int j,        /* which actuator 0-5                */
                          double actual, /* actual length of actuator j  (in) */
                          double temp)  /* temperature                 (deg_C) */
```

The following function is a place-holder; the actuator length calibration done by the metrology group [Par96] will be incorporated into this function in a future release of this code.

```
/* srActuatorsLengths.c --- compute actuator length corrections
   D.Wells, NRAO-CV
   1997-05-21: initial version
   1998-01-15: minor revisions
*/
```

[GNU GPL copyright notice omitted]

```
#include "srInclude.h"
#include "gbtOpticalConstants.h"

double srActuatorsLengths(              /* returns act j encoder length (in) */
                          int j,        /* which actuator 0-5                */
                          double actual, /* actual length of actuator j  (in) */
                          double temp)  /* temperature                (deg_C) */
{
    double
        encoder;          /* encoder value to produce desired actual length */
```

The six actuators of the GBT Gregorian subreflector have been measured in the NRAO-GB metrology laboratory in order to relate the length values indicated by their magnetostrictive encoders to their actual lengths from U-joint to U-joint. This function incorporates this calibration information. It takes the `actual` length which is desired and returns the encoder value to command to get that length.

The function interface as defined here makes no provision for an error return on illegal `j` or illegal `actual`; it is presumed that such errors will be caught elsewhere in the Gregorian focus tracking algorithms.

The `temp` argument will be used to include thermal expansion coefficients in the calculation of the actual lengths. (The actual lengths will be the nominal values, except for zero point and scale errors, at the `RIGGING_TEMP`, which is $21.1°C = 70°F$ for the GBT.) The magnetostrictive encoders are made of some type of stainless steel, whereas the rest of the encoder is made of a steel which is similar to the steel in the GBT structure.[3] This stainless steel has a thermal coefficient which is significantly different from the coefficient for the conventional steel. The resulting differential thermal expansion is just barely significant in the GBT.

```
    switch (j) {
        case (0): /* Y1 actuator */
            encoder = actual; break;
        case (1): /* Y2 actuator */
            encoder = actual; break;
        case (2): /* Y3 actuator */
            encoder = actual; break;
        case (3): /* X1 actuator */
            encoder = actual; break;
        case (4): /* X2 actuator */
            encoder = actual; break;
        case (5): /* Z actuator */
            encoder = actual; break;
        default:
            encoder = actual;
    }
  return (encoder);
}
```

[3] With the exception that the backup structure of the Gregorian subreflector is made of *aluminum*.

Table 2: Points used in derivatives computation

Name	Δx	Δy	Δz
Home	0.00	0.00	0.00
G-x-y-z	-9.49	-3.63	-4.00
G-x-y+z	-9.49	-3.63	4.00
G-x+y-z	-9.49	3.63	-4.00
G-x+y+z	-9.49	3.63	4.00
G+x-y-z	9.49	-3.63	-4.00
G+x-y+z	9.49	-3.63	4.00
G+x+y-z	9.49	3.63	-4.00
G+x+y+z	9.49	3.63	4.00
F-x-y-z	-9.49	-22.63	-0.83
F-x-y+z	-9.49	-22.63	0.83
F-x+y-z	-9.49	11.63	-0.83
F-x+y+z	-9.49	11.63	0.83
F+x-y-z	9.49	-22.63	-0.83
F+x-y+z	9.49	-22.63	0.83
F+x+y-z	9.49	11.63	-0.83
F+x+y+z	9.49	11.63	0.83

3 Test cases & timing measurements

The four testcase calculations and two timing measurements shown in Figure 3 were made on a SPARC system. Case [1] verifies the lengths of the actuators in the "home" position. Case [2] computes a pose with substantial translation and tilt displacements. In Case [3] the lengths computed in case [2] are inverted in 8 Newton-Raphson iterations to yield the original commanded pose, after having started the iteration with zero displacement. Case [4] demonstrates that only 3 iterations are needed if the the starting pose is near the correct answer.

4 Partials of translation/tilt motions *wrt* actuator motions

During preparation of a review [Wel98] of the motion requirements for the subreflector, there was a need to know how the velocity and acceleration limits on actuators L_i would translate to limits on ΔX_s, ΔY_s, ΔX_s, θ_{x_n}, θ_y and θ_z over the allowed envelope of displacements. A program was built to calculate the required partial derivatives by numerical differentiation. Table 2 gives the coordinates of the points in the envelope for which partials were computed; it extends the official envelope to a wider box in the region around the "home" position in order to support beam switching. Tables 3 through 8 display the computed partials. In each case we want to know the maximum absolute value of the partial, and this is shown at the end of each table. For each point we look for the largest partial derivative along the row and print it in bold font. The summary row at the end of each table shows which actuator will be the limiting case for each type of motion. The actuators which have the largest derivatives in Tables 3 through 5 (Δx, Δy, Δz cases) are obvious after examining the geometry in Figures 1 and 2, but the 0.836 maximum for L_6 for the "F-x-y-z" case in Table 6 surprised the author, who expected that L_2 and L_3 would always be the limiting cases (as they are for the "home" case). The L_1 result in Table 8 surprised the author at first, but it was obvious after examination of the x_s coordinates in Table 1 (the lever arm for L_1 is almost twice as large as for L_2 and L_3).

```
1998-01-21T18:49:24Z  <gibbon.cv> SunOS 5.5

 Kernel says CPU's clock rate is 150.0 MHz.
 Kernel says main memory's clock rate is 50.0 MHz.

 Sun-4 floating-point controller version 0 found.
 A Fujitsu/Ross RT620 hyperSPARC chip is available.
 FPU's frequency appears to be approximately 152.9 MHz.


% [1] srDisplacementToLength() - 'home position' case:
%               trans = [          0,          0,          0 ]
%               tilts = [          0,          0,          0 ]
%                 len = [    110.948,    110.981,    110.981
%                            127.242,    127.242,     61.578 ]
%              status = [          0 ]
        10000 executions in 240000 usec -->   24.0 usec/execution

% [2] srDisplacementToLength() - large displacement case:
%               trans = [         18,        -24,          1 ]
%               tilts = [         -1,       -0.3,        0.5 ]
%                 len = [    135.455,    135.654,    137.007
%                             151.84,    151.901,    75.4609 ]
%              status = [          0 ]

% [3] srLengthToDisplacement() - inversion of [2] with trans[]=tilts[]=0:
%                 len = [    135.455,    135.654,    137.007
%                             151.84,    151.901,    75.4609 ]
%               trans = [          0,          0,          0 ]
%               tilts = [          0,          0,          0 ]
%               trans = [         18,        -24,    1.00003 ]
%               tilts = [  -0.999988,  -0.299997,        0.5 ]
%              status = [          0 ]
%       8 Newton-Raphson iterations.

% [4] srLengthToDisplacement() - inversion of [2] with trans[]=tilts[]=+/-eps:
%                 len = [    135.455,    135.654,    137.007
%                             151.84,    151.901,    75.4609 ]
%               trans = [     18.001,    -24.001,    1.00103 ]
%               tilts = [   -1.00099,  -0.298997,      0.499 ]
%               trans = [         18,        -24,   0.999987 ]
%               tilts = [  -0.999965,  -0.300006,        0.5 ]
%              status = [          0 ]
%       3 Newton-Raphson iterations.
        10000 executions in 1100000 usec -->  110.0 usec/execution
```

Figure 3: Test cases and timing measurements

Table 3: Partials of length L_i *wrt* translation Δx $(\frac{\partial L_i}{\partial \Delta x})$

(inch/inch)

Point	L_1 Y1	L_2 Y2	L_3 Y3	L_4 X1	L_5 X2	L_6 Z1
Home	-0.010	-0.003	-0.003	**0.963**	**0.963**	-0.011
G-x-y-z	-0.092	-0.085	-0.085	**0.948**	**0.948**	-0.169
G-x-y+z	-0.092	-0.085	-0.085	**0.948**	**0.948**	-0.150
G-x+y-z	-0.098	-0.091	-0.091	**0.965**	**0.965**	-0.176
G-x+y+z	-0.098	-0.091	-0.091	**0.965**	**0.965**	-0.156
G+x-y-z	0.073	0.079	0.079	**0.961**	**0.961**	0.148
G+x-y+z	0.073	0.079	0.079	**0.961**	**0.961**	0.132
G+x+y-z	0.078	0.085	0.085	**0.974**	**0.974**	0.154
G+x+y+z	0.078	0.085	0.085	**0.974**	**0.974**	0.136
F-x-y-z	-0.079	-0.073	-0.073	**0.894**	**0.894**	-0.139
F-x-y+z	-0.079	-0.073	-0.073	**0.894**	**0.894**	-0.137
F-x+y-z	-0.106	-0.099	-0.099	**0.981**	**0.981**	-0.173
F-x+y+z	-0.106	-0.099	-0.099	**0.981**	**0.981**	-0.168
F+x-y-z	0.063	0.068	0.068	**0.919**	**0.919**	0.122
F+x-y+z	0.063	0.068	0.068	**0.919**	**0.919**	0.120
F+x+y-z	0.084	0.092	0.092	**0.986**	**0.986**	0.151
F+x+y+z	0.084	0.092	0.092	**0.986**	**0.986**	0.147
Limit case(s):				**0.986**	**0.986**	

Table 4: Partials of length L_i *wrt* translation Δy $(\frac{\partial L_i}{\partial \Delta y})$

(inch/inch)

Point	L_1 Y1	L_2 Y2	L_3 Y3	L_4 X1	L_5 X2	L_6 Z1
Home	**-1.000**	**-1.000**	**-1.000**	-0.268	-0.268	-0.347
G-x-y-z	**-0.995**	**-0.996**	**-0.996**	-0.316	-0.316	-0.416
G-x-y+z	**-0.995**	**-0.996**	**-0.996**	-0.316	-0.316	-0.371
G-x+y-z	**-0.994**	**-0.995**	**-0.995**	-0.260	-0.260	-0.309
G-x+y+z	**-0.994**	**-0.995**	**-0.995**	-0.260	-0.260	-0.273
G+x-y-z	**-0.997**	**-0.996**	**-0.996**	-0.274	-0.274	-0.418
G+x-y+z	**-0.997**	**-0.996**	**-0.996**	-0.274	-0.274	-0.372
G+x+y-z	**-0.996**	**-0.996**	**-0.996**	-0.224	-0.224	-0.310
G+x+y+z	**-0.996**	**-0.996**	**-0.996**	-0.224	-0.224	-0.274
F-x-y-z	**-0.997**	**-0.997**	**-0.997**	-0.448	-0.448	-0.606
F-x-y+z	**-0.997**	**-0.997**	**-0.997**	-0.448	-0.448	-0.595
F-x+y-z	**-0.994**	**-0.995**	**-0.995**	-0.194	-0.194	-0.166
F-x+y+z	**-0.994**	**-0.995**	**-0.995**	-0.194	-0.194	-0.162
F+x-y-z	**-0.998**	**-0.998**	**-0.998**	-0.394	-0.394	-0.607
F+x-y+z	**-0.998**	**-0.998**	**-0.998**	-0.394	-0.394	-0.597
F+x+y-z	**-0.996**	**-0.996**	**-0.996**	-0.167	-0.167	-0.167
F+x+y+z	**-0.996**	**-0.996**	**-0.996**	-0.167	-0.167	-0.163
Limit case(s):	**-1.000**	**-1.000**	**-1.000**			

Table 5: Partials of length L_i *wrt* translation Δz $(\frac{\partial L_j}{\partial \Delta z})$

(inch/inch)

Point	L_1 Y1	L_2 Y2	L_3 Y3	L_4 X1	L_5 X2	L_6 Z1
Home	0.000	0.000	0.000	0.000	0.000	**0.937**
G-x-y-z	-0.034	-0.034	-0.034	-0.033	-0.033	**0.893**
G-x-y+z	0.035	0.035	0.035	0.034	0.034	**0.916**
G-x+y-z	-0.037	-0.037	-0.037	-0.034	-0.034	**0.934**
G-x+y+z	0.038	0.038	0.038	0.035	0.035	**0.949**
G+x-y-z	-0.034	-0.034	-0.034	-0.029	-0.029	**0.897**
G+x-y+z	0.035	0.035	0.035	0.029	0.029	**0.919**
G+x+y-z	-0.037	-0.037	-0.037	-0.029	-0.029	**0.938**
G+x+y+z	0.038	0.038	0.038	0.030	0.030	**0.952**
F-x-y-z	-0.006	-0.006	-0.006	-0.006	-0.006	**0.783**
F-x-y+z	0.007	0.007	0.007	0.007	0.007	**0.792**
F-x+y-z	-0.008	-0.008	-0.008	-0.007	-0.007	**0.971**
F-x+y+z	0.009	0.009	0.009	0.008	0.008	**0.972**
F+x-y-z	-0.006	-0.006	-0.006	-0.005	-0.005	**0.785**
F+x-y+z	0.007	0.007	0.007	0.006	0.006	**0.794**
F+x+y-z	-0.008	-0.008	-0.008	-0.006	-0.006	**0.974**
F+x+y+z	0.009	0.009	0.009	0.007	0.007	**0.976**
Limit case(s):						**0.976**

Table 6: Partials of length L_i *wrt* tilt θ_{x_n} $(\frac{\partial L_j}{\partial \theta_{x_n}})$

(inch/degree)

Point	L_1 Y1	L_2 Y2	L_3 Y3	L_4 X1	L_5 X2	L_6 Z1
Home	0.000	**0.717**	**-0.717**	-0.322	0.322	0.687
G-x-y-z	-0.040	0.742	-0.774	-0.288	0.271	**0.784**
G-x-y+z	0.040	**0.774**	-0.742	-0.271	0.288	0.758
G-x+y-z	-0.042	0.744	**-0.778**	-0.337	0.320	0.739
G-x+y+z	0.043	**0.778**	-0.743	-0.320	0.337	0.714
G+x-y-z	-0.040	0.654	**-0.687**	-0.323	0.309	0.636
G+x-y+z	0.040	**0.687**	-0.654	-0.309	0.324	0.626
G+x+y-z	-0.042	0.650	**-0.685**	-0.366	0.351	0.585
G+x+y+z	0.043	**0.685**	-0.650	-0.351	0.366	0.578
F-x-y-z	-0.007	0.750	-0.756	-0.158	0.154	**0.836**
F-x-y+z	0.007	0.756	-0.750	-0.154	0.158	**0.833**
F-x+y-z	-0.009	0.761	**-0.768**	-0.385	0.382	0.665
F-x+y+z	0.010	**0.769**	-0.761	-0.382	0.385	0.661
F+x-y-z	-0.007	0.675	-0.680	-0.209	0.206	**0.714**
F+x-y+z	0.007	0.680	-0.675	-0.206	0.209	**0.713**
F+x+y-z	-0.009	0.660	**-0.668**	-0.407	0.404	0.514
F+x+y+z	0.010	**0.668**	-0.660	-0.404	0.407	0.513
Limit case(s):						**0.836**

Table 7: Partials of length L_i *wrt* tilt θ_y $(\frac{\partial L_j}{\partial \theta_y})$

(inch/degree)

Point	L_1 Y1	L_2 Y2	L_3 Y3	L_4 X1	L_5 X2	L_6 Z1
Home	0.000	-0.003	0.003	**0.860**	**-0.860**	0.294
G-x-y-z	0.035	-0.095	0.058	0.822	**-0.870**	0.154
G-x-y+z	-0.035	-0.058	0.095	**0.870**	-0.822	0.176
G-x+y-z	0.037	-0.101	0.062	0.837	**-0.885**	0.161
G-x+y+z	-0.037	-0.062	0.101	**0.885**	-0.837	0.182
G+x-y-z	0.035	0.052	-0.089	0.837	**-0.878**	0.406
G+x-y+z	-0.035	0.089	-0.052	**0.878**	-0.837	0.400
G+x+y-z	0.037	0.056	-0.095	0.848	**-0.890**	0.425
G+x+y+z	-0.037	0.095	-0.056	**0.890**	-0.848	0.415
F-x-y-z	0.006	-0.069	0.063	0.793	**-0.802**	0.141
F-x-y+z	-0.006	-0.063	0.069	**0.802**	-0.793	0.146
F-x+y-z	0.008	-0.093	0.084	0.870	**-0.880**	0.175
F-x+y+z	-0.008	-0.084	0.093	**0.880**	-0.870	0.180
F+x-y-z	0.006	0.057	-0.064	0.816	**-0.824**	0.350
F+x-y+z	-0.006	0.064	-0.057	**0.824**	-0.816	0.351
F+x+y-z	0.008	0.077	-0.086	0.875	**-0.884**	0.434
F+x+y+z	-0.008	0.086	-0.077	**0.884**	-0.875	0.431
Limit case(s):				**0.890**	**-0.890**	

Table 8: Partials of length L_i *wrt* tilt θ_z $(\frac{\partial L_j}{\partial \theta_z})$

(inch/degree)

Point	L_1 Y1	L_2 Y2	L_3 Y3	L_4 X1	L_5 X2	L_6 Z1
Home	**-0.991**	0.530	0.530	-0.624	-0.624	0.122
G-x-y-z	**-0.930**	0.608	0.608	-0.577	-0.577	0.279
G-x-y+z	**-0.930**	0.608	0.608	-0.577	-0.577	0.249
G-x+y-z	**-0.926**	0.613	0.613	-0.631	-0.631	0.251
G-x+y+z	**-0.926**	0.613	0.613	-0.631	-0.631	0.222
G+x-y-z	**-1.045**	0.448	0.448	-0.617	-0.617	0.010
G+x-y+z	**-1.045**	0.448	0.448	-0.617	-0.617	0.009
G+x+y-z	**-1.048**	0.442	0.442	-0.663	-0.663	-0.030
G+x+y+z	**-1.048**	0.442	0.442	-0.663	-0.663	-0.027
F-x-y-z	**-0.941**	0.597	0.597	-0.438	-0.438	0.315
F-x-y+z	**-0.941**	0.597	0.597	-0.438	-0.438	0.310
F-x+y-z	**-0.920**	0.620	0.620	-0.690	-0.690	0.202
F-x+y+z	**-0.920**	0.620	0.620	-0.690	-0.690	0.197
F+x-y-z	**-1.039**	0.459	0.459	-0.497	-0.497	0.093
F+x-y+z	**-1.039**	0.459	0.459	-0.497	-0.497	0.091
F+x+y-z	**-1.052**	0.436	0.436	-0.714	-0.714	-0.074
F+x+y+z	**-1.052**	0.436	0.436	-0.714	-0.714	-0.072
Limit case(s):	**-1.052**					

5 Safety Considerations

In principle it is possible to command a combination of actuator lengths which will cause the mechanical parts of the system to bind or to collide with each other or with other elements of the telescope. Some combinations could bend one or more U-joints excessively, endangering the rubber seals. The design engineers studied these possibilities and their summary statement [Zai92, p.5] was: "*There is no danger of exceeding hardware limits if the subreflector operates within its provided envelope* and *none of the 'extreme deltas' are exceeded.*" These conclusions are implemented by a set of multi-level tests, both software and hardware, which are discussed in [Zai92].

A key concept is a set of six rules concerned with differences between actuator lengths. These rules appear in the last ten lines of code of `srDisplacementToLength()` (Section 2.1). The actuators have potentiometers which produce analog voltages proportional to encoder readings, and these are combined in operational amplifier and comparator circuits to provide an analog hardware implementation of these rules, with slightly looser limits than in the software. If these hardware limit detectors trigger, they disable the system and necessitate a manual reset. The PCD control software for the actuator servos also implements these rules in software, and inhibits motions when the rules are violated, which should prevent triggering the hardware limits.

The six length-difference rules are a necessary but not sufficient condition for the safety of the system: we must also limit displacement to combinations inside a specified box. In [Zai92] the corners of this box are specified as the eight points labelled "$F \pm x \pm y \pm z$" in Table 2.

Motions in Δz larger than ± 0.83 inch may be desirable in order to move the beam between feedhorns separated by up to about 14 inches in the Gregorian focal plane while maintaining optimal imaging quality. Probably the best way to implement this will be to modify PCD's firmware implementation so that it will permit displacements inside the box defined by the eight points labelled "$G \pm x \pm y \pm z$" in Table 2 in addition to permitting displacements inside the $F \pm x \pm y \pm z$ box.

A srInclude.h

The error message macros which appear below are experimental—it is not yet clear whether they will be useful in the M&C implementation.

```
/* srInclude.h -- Include for the GBT's Gregorian subreflector
   D.Wells, NRAO-CV
   1996-02-14: new version of sr actuator include
   1997-04-25: srInclude merges ellipsoid.h,greg_focus_track.h,greg_actuators.h
   1997-05-21: eliminated enum gft_mode
   1997-06-13: change made to test procedure for moving code to M&C
   1997-07-23: added macro GET_SR_TILTS()
   1998-01-15: removed focus tracking debug code & Schwab iteration macro
*/
```

[GNU GPL copyright notice omitted]

```
#ifndef SR_INCLUDE_H
#define SR_INCLUDE_H

#include <math.h>
#include "smInclude.h"

/* -=- Macros to define actuator names and length-rule error messages: -=- */
#define SRL0 Y1
#define SRL1 Y2
#define SRL2 Y3
#define SRL3 X1
#define SRL4 X2
#define SRL5 Z1
#define SR_ACTUATORS_NAMES_N(A) #A
#define SR_ACTUATORS_ERRORS_M(A,B,C,D) A "(" #B " - " #C ") " D
#define SR_ACTUATORS_NAMES_A(A0,A1,A2,A3,A4,A5) \
    char *srActuatorsNames[] = {SR_ACTUATORS_NAMES_N(A0), \
                                SR_ACTUATORS_NAMES_N(A1), \
                                SR_ACTUATORS_NAMES_N(A2), \
                                SR_ACTUATORS_NAMES_N(A3), \
                                SR_ACTUATORS_NAMES_N(A4), \
                                SR_ACTUATORS_NAMES_N(A5)}
#define SR_ACTUATORS_NAMES SR_ACTUATORS_NAMES_A(SRL0,SRL1,SRL2,SRL3,SRL4,SRL5)
#define SR_ACTUATORS_ERRORS_A(A0,A1,A2,A3,A4,A5)  \
    struct srActuatorsErrorStruct {         \
        int bit;                            \
        char *message;                      \
    } srActuatorsLengthErrors[] = {         \
        { 1, SR_ACTUATORS_ERRORS_M("abs", A0, A2, ">  2.60")}, \
        { 2, SR_ACTUATORS_ERRORS_M("abs", A0, A2, ">  2.60")}, \
        { 4, SR_ACTUATORS_ERRORS_M("abs", A1, A2, ">  2.62")}, \
        { 8, SR_ACTUATORS_ERRORS_M("abs", A3, A4, ">  1.40")}, \
        {16, SR_ACTUATORS_ERRORS_M(   "", A1, A5, "< 39.19")}, \
        {32, SR_ACTUATORS_ERRORS_M(   "", A1, A5, "> 62.98")}  \
    }; \
    const int NsrActuatorsErrors = (sizeof(srActuatorsLengthErrors)   \
                                    / sizeof(struct srActuatorsErrorStruct))
#define SR_ACTUATORS_ERRORS SR_ACTUATORS_ERRORS_A(SRL0,SRL1,SRL2,SRL3,SRL4,SRL5)
```

```
/* -=-=-=-=-=-=-=- Function Prototypes: -=-=-=-=-=-=-=- */
#ifdef __cplusplus
extern "C" {
#endif /* endif cplusplus */

int srDisplacementToLength(                  /* returns !=0 on delta-errors  */
                           double trans[3],  /* delta translations      (in) */
                           double tilts[3],  /* delta tilts            (deg) */
                           double temp,      /* temperature          (deg_C) */
                           double encoder[6], /* output: lengths to use  (in) */
                           double apex_delta[3][6],     /* U-joint errors(in) */
                           double home_sub_delta[3][6]) /* U-joint errors(in) */
;
int srLengthToDisplacement(                 /* returns !=0 if maxdelta fails */
                           double len[6],   /* input actuator lengths   (in) */
                           double temp,     /* temperature          (deg_C) */
                           double trans[3], /* delta translations       (in) */
                           double tilts[3], /* delta tilts             (deg) */
                           double apex_delta[3][6],      /* U-joint error(in) */
                           double home_sub_delta[3][6],  /* U-joint error(in) */
                           int    *iter)                 /* N-R iterations    */
;
double srActuatorsLengths(              /* returns act j encoder length (in) */
                          int j,        /* which actuator 0-5                */
                          double actual, /* actuallength of act j        (in) */
                          double temp)  /* temperature              (deg_C) */
;
void srEllipsoid (double s12,    /* Separation=(F1F2-11m) (m) */
                  double t,      /* Axis-tilt       (radians) */
                  double dxyc[]) /* Center_offset         (m) */
;
double srEllipsoidBestRms (             /* returns extra_tilt=dphi     (mr) */
                           double ds12) /* delta_separation=(F1F2-11m) (mm) */
;
int srFocusTracking (                    /* returns != 0 on error      */
                     double elev,        /* elevation            (deg) */
                     double temp,        /* temperature        (deg_C) */
                     double F1_delta[3], /* first-focus offset     (m) */
                     double F2_delta[3], /* second-focus offset    (m) */
                     double sub_trans[], /* XYZ subrefl backup    (in) */
                     double sub_tilts[], /* xyz subrefl tilts    (deg) */
                     struct node_data
                            *greg_feed,  /* feed coords/tilts wrt BFP  */
                     double *dL12,       /* PFP->feed - 11m        (m) */
                     double *extra_tilt, /* extra tilt subrefl   (rad) */
                     double *delta_tilt, /*                      (rad) */
                     double S[],         /* subrefl opticl vertex (in) */
                     double E[])         /* Euler ang vertex tilt(rad) */
;

#ifdef __cplusplus
};
#endif /* endif cplusplus */
#endif /* SR_INCLUDE_H */
```

References

[Kin94] Lee King. GBT coordinate systems. Limited-distribution memo, January 1994.

[KM93] Lee King and Greg Morris. Foci arrangement and coordinate systems for the GBT. GBT Drawing C35102M081, NRAO, December 1993. The first sheet of this set of five drawings schematically defines six different coordinate systems to be used in the GBT project. Sheets 2-5 define the algebraic relationships between these coordinate systems.

[Nan91] P. B. Nanavati. Equations of motion – subreflector positioner. Loral GBT Technical Memo 19, Loral Western Development Labs, September 1991. The "inverse kinematic" algorithm for calculating actuator lengths from displacements is derived by both matrix/vector and trigonometric approaches. Appendix A includes a copy of NRAO drawing B35102M007 which specifies the "nutation" axis X_n.

[Par96] David H. Parker. Subreflector actuator calibrations. Cover memo addressed to Bob Hall, with reports "P0040" and "P0041" attached. P0040 is from M. A. Goldman 1996-01-21 and is titled "Determination of Actuator Length for the Secondary Reflector of the Green Bank Telescope". P0041 is from Bill Radcliff and David Bradley to David Parker 1996-02-23 and is titled "GBT Secondary Reflector Actuator End-to-End Length Measurements"., February 1996.

[Sch92] F. Schwab. Error sensitivity of the GBT subreflector positioning mechanism. GBT Memo 93, NRAO, November 1992. The memo addresses the question "What is the nature of the subreflector positioning errors that would result from imperfect behavior of the U-joints and ball screws of the subreflector positioning mechanism?".

[Sri90] S. Srikanth. Axial focussing. GBT Memo 49, NRAO, April 1990. "..the axis along which the [subreflector is] to be moved (without the telescope beam shifting in the sky) for axial focussing and the amount of movement required [are] determined..; the axis of focussing is 36.73° from the axis of the.. main reflector.. [and] 24.4° from the secondary focus feed axis..".

[Ste66] D. Stewart. A platform with six degrees of freedom. *Proc. Inst. Mech. Engrs., Part I*, 180(15):371–386, 1965–66.

[Wel98] Don Wells. GBT subreflector motions and servo properties. GBT Memo 176, NRAO, January 1998. Five types of GBT subreflector motions needed for astronomical observations are discussed, and estimates of their amplitudes and frequencies are given. These estimates are translated into velocity and acceleration requirements, which are compared with the specifications of the existing servo implementation.

[WK95] Don Wells and Lee King. The GBT Tipping-Structure Model in C. GBT Memo 124, NRAO, March 1995. Abstract: The finite element model of the GBT tipping structure has been translated into executable code expressed in the C language, so that it can be used by the control software modules for the pointing, focus-tracking, quadrant detector, active-surface and laser-rangefinder subsystems of the GBT. We give a description of this C-code version of the tipping structure model and two examples of its application to practical problems. See `ftp://fits.cv.nrao.edu/pub/gbt_dwells_doc.tar.gz` for the current revision of this memo (124.3 as of 1997-06-23).

[Zai92] J. Zaine. Mechanical analysis S/R positioner. Loral GBT Technical Memo 46, Loral Western Development Labs, October 1992. This comprehensive report discusses operational, peak operational and survival loadings, required motor torques, lost motion (backlash) of the U-joints and possible structural interference and U-joint angle problems of the subreflector actuators. Portions of Loral Interoffice Memorandum 3WL110-DLE-107, titled "NRAO 100m GBT Prime Focus Feed & Subreflector Positioner Gravity Correction Travel Requirements" are included as Appendix A (pp.A1-A13). The asymmetric structural model which was used enabled calculation of the required out-of-plane Z_s translation ±0.83in.